ADVANCED TELEPROCESSING SYSTEMS
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

FINAL TECHNICAL REPORT

September 30, 1987

Principal Investigator: Leonard Kleinrock

Computer Science Department
School of Engineering and Applied Science
University of California
Los Angeles

ADVANCED TELEPROCESSING SYSTEMS
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

FINAL TECHNICAL REPORT

September 30, 1987

Principal Investigator: Leonard Kleinrock

Computer Science Department
School of Engineering and Applied Science
University of California
Los Angeles

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A187 516	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ADVANCED TELEPROCESSING SYSTEMS: Final Technical Report	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 10/1/81 - 9/30/87
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Leonard Kleinrock	8. CONTRACT OR GRANT NUMBER(s) MDA 903-82-C-0064
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Engineering and Applied Science University of California, Los Angeles Los Angeles, California 90024-1596	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DARPA Order No. 2496
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	12. REPORT DATE September 30, 1987 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: Distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Please see next page.

DTIC COPY INSPECTED 6

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This Final Technical Report covers research carried out by the Advanced Teleprocessing Systems at UCLA under contract MDA- C-82-0064 from the period October 1, 1981 to September 30, 1987.

This contract has had three primary designated research areas: Distributed Communications Access; Distributed Processing; Distributed Control and Algorithms. This report briefly summarizes our research results and accomplishments in these areas during this period.

Under our first task, distributed communications access, we conducted an indepth research effort in evaluating the performance of packet radio systems in a variety of environments. We pushed the analytic side of this field to its point of diminishing returns and found that we had reached a barrier beyond which the analytic payoff will be meager and the analytic effort would be enormous. We have determined a number of key principles and properties of multi-hop packet radio networks and have reported upon these in our publications. The problem of a complete analysis of multi-hop packet radio systems was found to be intractable and, therefore, we phased out that portion of the research half way through this contract. We devoted much of our effort to the final two tasks, namely, distributed processing and distributed control and algorithms. Here, we have launched into uncharted terrain in an attempt to uncover the proper models with would expose the important principles of Distributed Systems. Our efforts here have been rewarded with some important successes. We have studied distributed algorithms in networks involving such functions as elections and traversal. Local area networks has been an object of our study and here, the introduction of a broadcast communications environment has provided the motivation for some rather extensive evaluation of broadcast communications. We have found there are some important advantages of broadcast communications in a number of these environments. We have begun to lay down the foundation of a theory of distributed processing. Furthermore, parallel processing systems have yielded in some important ways to our analysis and we have been able to get a very general result regarding the concurrency in a very general parallel processing system. Load balancing has been an object of importance to us, and here, we have invented some useful load balancing algorithms and have provided an evaluation of their performance. Extensions to the basic analytic techniques for many of these systems were published not only in our papers, but also in some books that were completed.

We provide a complete list of publications in this final technical report listing journal publications, Ph.D. dissertations, Master's thesis, and books published.

In the final section of this report, we reproduced two papers published by the Principal Investigator (Leonard Kleinrock). The first of these is, "Distributed System", and this was published in *ACM/IEEE-CS* Joint Special Issue, (November 1985): *Communications of the ACM*, Vol. 28, No. 11, pp. 1200-1213, and *Computer*, Vol. 18, No. 11, pp. 90-103; the second of these is, "Performance Models For Distributed Systems", and this was published in *Teletraffic Analysis and Computer Performance Evaluation*, the Proceedings of the International Seminar, 1986, pp. 1-16. These papers provide an indication as to some of our results, but more importantly, an indication as where some of the open and remaining important problems are which define the thrust of our continued studies.

ADVANCED TELEPROCESSING SYSTEMS

Final Technical Report

September 30, 1987

Contract Number: MDA 903-82-C-0064
DARPA Order Number: 2496
Contract Period: October 1, 1981 to September 30, 1987
Report Period: October 1, 1981 to September 30, 1987

Principal Investigator: Leonard Kleinrock

(213) 825-2543

Computer Science Department
School of Engineering and Applied Science
University of California, Los Angeles

Sponsored by

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

ADVANCED TELEPROCESSING SYSTEMS

Defense Advanced Research Projects Agency
Final Technical Report

September 30, 1987

This Final Technical Report covers research carried out by the Advanced Teleprocessing Systems Group at UCLA under DARPA Contract No. MDA 903-82-C-0064 covering the period from October 1, 1981 to September 31, 1987. We restate below the tasks which have been the subject of this research project. That is followed by an extensive bibliography of those published papers which have appeared during the contract period from October 1, 1981 through September 30, 1987. Although this contract was formally ended on September 30, 1987, the period from July 1, 1986 through September 30, 1987 was simply a no cost extension of funds to complete and publish results of studies that had begun during the main contract period. At the end of this report, we reproduce two papers entitled, "Distributed Systems" and "Performance Models For Distributed Systems", which were authored by the Principal Investigator (Leonard Kleinrock) and published as invited papers for ACM/IEEE-CS Joint Special Issue (November 1985): *Communications of the ACM*, Vol. 28, No. 11, pp. 1200-1213, and *Computer*, Vol. 18, No. 11, pp. 90-103; and *Teletraffic Analysis And Computer Performance Evaluation*, the Proceedings of the International Seminar in Amsterdam, June 1986, pp. 1-16, respectively. These papers summarize many of the research activities and results of this contract. The research conducted earlier in this period has been amply reported upon in our semi-annual technical reports as well as in the published professional literature (see the bibliography below).

During this contract period, we have been engaged in the following three designated tasks:

TASK I. DISTRIBUTED COMMUNICATIONS ACCESS

> The general problem of sharing a multi-access broadcast distributed systems among a set of competing users will be studied. General issues involving exhaustive communications, start-up problems and refined models to manifest some more realistic phenomena in these systems will be studied. Applications to packet radio systems and large survivable networks involving the study of tandem networks, multi-hop net-

works, one-way communication links, correct reception of more than one simultaneous transmission and mobility will be included. Further applications will include the study of very high bandwidth channels and/or very long propagation delay systems, multiple token systems and compound hierarchical network structures.

TASK II. DISTRIBUTED PROCESSING

The interplay between distributed communications in a broadcast environment and processing of distributed data will be studied. For example, the effect of merging sorted lists in a broadcast environment, as well as finding properties of elements in these lists, will be studied. Concurrency in multiprocessor systems will be studied in order to investigate performance in terms of response time and speedup factors for various graph models of computation. Connection architectures for multiprocessor systems will be investigated as well. One application here is the structure of the processing and communication architecture for supercomputers.

TASK III. DISTRIBUTED CONTROL AND ALGORITHMS

Routing, flow control and survivability in large packet radio networks as well as in public data networks will be studied as control algorithms in a distributed environment. Measures of performance, including throughput, response time, blocking, power, fairness, and robustness will be applied to these systems. Distributed algorithms for finding shortest paths, connectivity, loops, etc. will be studied. The effect of node and link failures, limited amounts of memory at each node and restricted channel capacity for communications will be investigated. The effect of network failures and delays on distributed data base management systems will also be studied.

These tasks have advanced in some major ways during this research period. The output of the research effort during the period October 1, 1981 through September 30, 1987, has resulted in the completion of:

a. 8 PhD. dissertations,

b. 51 publications by the Principal Investigator, faculty, staff and students appearing in the professional literature,

c. 5 M.S. theses.

and a number of accepted and submitted papers (all of which are pending publication), which have not yet appeared in the published literature. These publications are listed in the bibliography below.

Thus, as one measure of our achievements during this contract period, we offer the publications and Ph.D. professionals listed below. In addition to these, there has been a large number of presentations at professional conferences around the world. The substance of recent progress in distributed systems is to be found in the papers following the publication list.

ATS PUBLICATIONS
DARPA CONTRACT

Computer Science Department
University of California, Los Angeles
Professor Leonard Kleinrock

October 1981 - September 1987

BOOKS PUBLISHED

1. Kleinrock, L. and R. Gail, *Solutions Manual for Queueing Systems, Vol. I: Theory*, Technology Transfer Institute, Los Angeles, 1982.

2. Kleinrock, L. and R. Gail, *Solutions Manual for Queueing Systems, Vol. II: Computer Applications*, Technology Transfer Institute, Los Angeles, 1986.

CHAPTERS IN BOOKS

1. Gerla, M. and L. Kleinrock, "Flow Control Protocols", in *Computer Network Architectures and Protocols*, Green, Paul E., J. (ed.), Plenum Press, New York and London, 1982, pp. 361-412.

2. Kleinrock, L., "Channel Efficiency for LANs", in *Local Area & Multiple Access Networks*, Pickholtz R. (ed.), Computer Science Press, Rockville, Maryland, 1986, pp. 31-41.

3. Kleinrock, L., "Performance Models for Distributed Systems", invited paper (and keynote presentation) for *Teletraffic Analysis and Computer Performance Evaluation*, the Proceedings of the International Seminar held at The Centre for Mathematics and Computer Science June 2-6, 1986, Amsterdam; North-Holland, Amsterdam, June 1986.

Ph.D DISSERTATIONS

1. Nelson, Randolph "Channel Access Protocols for Multi-hop Broadcast Packet Radio Networks', July 1982.

2. Takagi, Hideaki "Analysis of Throughput and Delay for Single and Multi-hop Packet Radio Networks", May 1983.

3. Sadr, Ramin "UCLA Demodulation Engine", May 1983.

4. Gail, H. R. "On the Optimization of Computer Network Power," September 1983.

5. Levy, H. "Non-Uniform Structures and Synchronization Patterns in Shared Channel Communication Networks," August 1984.

6. Kung, K. C. "Concurrency in Parallel Processing Systems," June 1984.

7. Afek, Y. "Distributed Algorithms for Election in Unidirectional and Complete Networks," November 1985.

8. Belghith, A. "Response Time and Parallelism in Parallel Processing Systems with Certain Synchronization Constraints," available as UCLA, Computer Science Department Report No. 860015, December 1986.

M. S. THESES

1. Rosenberg, C. P. "Exponential Queueing Systems with Randomly Changing Arrival and Service Rates," 1984.

2. Green, J. J. "Analysis of Time Stamp Queue," 1984.

3. Korfhage, W. R. "Distributed Election in Unidirectional Eulerian Networks, with Application," March 1985.

4. Huang, J. H. "Throughput Analysis for Certain Multi-access Protocols with Imperfect Sensing," March 1985.

5. Felderman, R. E. "Flocks of Birds as a Paradigm for Distributed Systems," 1986.

PUBLISHED RESEARCH PAPERS

1. Kleinrock, L. "A Decade of Network Development", *Journal of Telecommunication Networks*, Spring 1982, pp. 1-11.

2. Kleinrock, L. "Packet Switching Principles", *Proceedings of the L. M. Ericsson Award Ceremony*, Stockholm, Sweden, May 5, 1982, also as a special editorial in *Journal of Telecommunication Networks*, Spring 1983, as well as in *Ericsson Reviews*.

3. Nelson, R. and L. Kleinrock, "Mean Message Delay in Multi-hop Packet Radio Networks Using Spatial-TDMA", *Conference Record, International Conference on Communications*, Philadelphia, June 13-17, 1982, pp. 1C.4.1-1C.4.4.

4. Naylor, W. and L. Kleinrock, "Stream Communication in Packet-Switched Networks", *IEEE Transactions on Communications*, December 1982, pp. 2527-2534.

5. Belghith, A. and L. Kleinrock, "Distributed Routing Scheme in Stationless Multi-hop Packet Radio Networks 'Mobility Handling'", *ACM SIGCOMM '83 Proceedings*, March 9, 1983, Austin, Texas, pp. 101-108; Packet Radio Temporary Note #313.

6. Nelson, R. and L. Kleinrock, "Maximum Probability of Successful Transmission in a Random Planar Packet Radio Network', *IEEE Infocom '83 Proceedings*, April 18-21, 1983, San Diego, California, pp. 365-370.

7. Silvester, J. and L. Kleinrock, "On the Capacity of Single-Hop Slotted ALOHA Networks for Various Traffic Matrices and Transmission Strategies," *IEEE Transactions on Communications*, Vol. COM-31, No. 8, August 1983. pp. 983-991.

8. Silvester, J. and L. Kleinrock, "On the Capacity of Multihop Slotted ALOHA Networks with Regular Structure," *IEEE Transactions on Communications*, Vol. COM-31, No. 8, August 1983, pp. 974-982.

9. Scholl, M. and L. Kleinrock, "On the M/G/1 Queue with Rest Periods and Certain Service Independent Queueing Disciplines," *Operations Research*, Vol. 31, No. 4, July-August 1983, pp. 705-719.

10. Gerla, M., Kleinrock, L. and Y. Afek, "A Distributed Routing Algorithm for Unidirectional Networks," *IEEE Global Telecommunications Conference* GLOBECOM '83, November 28-December 1, 1983, San Diego, CA.

11. Gail, H. R., "On the Optimization of Computer Network Power," UCLA, Computer Science Department Report No. 830922, 1983.

12. Kleinrock, L. and G. Akavia, "On a Self-Adjusting Capability of ALOHA Networks," *IEEE Transactions on Communications*, Vol. COM-32, No. 1, January 1984, pp. 40-47.

13. Takagi, H. and L. Kleinrock, "Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals," *IEEE Transactions on Communications*, Vol. COM-32, No. 3, March 1984, pp. 246-257.

14. Takagi, H. and L. Kleinrock, "Diffusion Process Approximation for the Queueing Delay in Contention Packet Broadcasting Systems," *Performance of Computer Communication Systems*, Proceedings of the International Symposium held in Zurich, March 21-23, 1984, pp. 111-124.

15. Rodrigues, A. P., Fratta, L. and M. Gerla, "Token-less Protocols for Fiber Optics Local Area Networks," *Proceedings of ICC '84*, Vol. 3, Amsterdam, May 1984, pp. 1150-1153.

16. Nelson, R. and L. Kleinrock, "The Spatial Capacity of a Slotted ALOHA Multihop Packet Radio Network with Capture," *IEEE Transactions on Communications*, Vol. COM-32, No. 6, June 1984, pp. 684-694.

17. Gafni, E. and Y. Afek, "Election and Traversal in Unidirectional Networks," *ACM Symposium Proceedings*, Principles of Distributed Computing, Vancouver, British Columbia, August 1984, and UCLA, Computer Science Department Report No. 850008.

18. Gafni, E., Afek, Y. and L. Kleinrock, "Fast and Message-Optimal Synchronous Election Algorithm for Complete Networks," UCLA, Computer Science Department Report No. 840041, October 1984.

19. Kleinrock, L., "On the Theory of Distributed Processing," presented at and published in the *Proceedings of the Twenty-Second Annual Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, October 1984, pp. 60-70.

20. Afek, Y and E. Gafni, "Simple and Efficient Distributed Algorithms for Election in Complete Networks," presented at and published in the *Proceedings of the Twenty-Second Annual Allerton Conference on* Communication, Control, and Computing, Urbana-Champaign, October 1984, pp. 689-698.

21. Gafni, E. and W. Korfhage, "Distributed Election in Unidirectional Eulerian Networks," presented at and published in the *Proceedings of the Twenty-Second Annual Allerton Conference on* Communication, Control, and Computing, Urbana-Champaign, October 1984, 699-700.

22. Afek, Y. and E. Gafni, "Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks," UCLA, Computer Science Department Report No. 850003, January 1985, also submitted to the *Proceedings* of *ACM* Sponsored Conference, 1985.

23. Takagi, Hideaki and L. Kleinrock, "A Tutorial on the Analysis of Polling Systems," UCLA, Computer Science Department Report No. 850005, February 1985.

24. Kleinrock, L., "On Queueing Problems in Random Access Communications," invited paper for *IEEE Transactions on Information Theory*, Special Issue, Vol. IT-31, No. 2, March 1985, pp. 166-175.

25. Gerla, M., Chan, H.W. and J.R. Boisson de Marca, "Fairness in Computer Networks," *Proceedings, IEEE International Conference on Communications*, (Chicago, June 23-26, 1985), Vol. 3, 1985, pp. 1384-1389.

26. Takagi, H. and L. Kleinrock, "Throughput Analysis for Persistent CSMA Systems," *IEEE Transactions on Communications*, Vol. COM-33, No. 7, July 1985, pp. 627-638.

27. Dechter, R. and L. Kleinrock, "Parallel Algorithms for Multiprocessors Using Broadcast Channel", Computer Science Department, UCLA Report No. 850025, January 1985 (formerly Working Paper No. 81002, November 30, 1981).

28 Gerla, M., "Routing and Flow Control in ISDN's" *ICCC '86 Proceedings*, Munich, Germany, September 1986.

29. Belghith, A. and L. Kleinrock, "Analysis of the Number of Occupied Processors in a Multiprocessing System," UCLA, Computer Science Department Report No. 850027, August 1985; revised and reissued as UCLA, Computer Science Department Report No. 870003, February 1987.

30. Nelson, R. and L. Kleinrock, "Rude CSMA: A Multihop Channel Access Protocol," *IEEE Transactions on Communications*, Vol. COM-33, No. 8, August 1985, pp. 785-791.

31. Nelson, R. and L. Kleinrock, "Spatial TDMA: A Collision Free Multihop Channel Access Protocol," *IEEE Transactions on Communications*, Vol. COM-33, No. 9, September 1985, pp. 934-944.

32. Molle, M. L. and L. Kleinrock, "Virtual Time CSMA: Why Two Clocks are Better Than One," *IEEE Transactions on Communications*, Vol. COM-33, No. 9, September 1985, pp. 919-933.

33. Gerla, M. and H.W. Chan, "Window Selection in Flow Controlled Networks," *Proceedings of the Ninth Data Communications Symposium*, Vancouver, Canada, September 1985, pp. 84-92.

34. Takagi, H. and L. Kleinrock, "Mean Packet Queueing Delay in a Buffered Two User CSMA/CD System," Correspondence in *IEEE Transactions on Communications*, Vol. COM-33, No. 10, October 1985, pp. 1136-1139.

35. Takagi, H. and L. Kleinrock, "Throughput-Delay Characteristics of Some Slotted-ALOHA Multihop Packet Radio Networks," *IEEE Transactions on Communications*, Vol. COM-33, No. 11, November 1985, pp. 1200-1207.

36. Takagi, H. and L. Kleinrock, "Output Processes in Contention Packet Broadcasting Systems," *IEEE Transactions on Communications*, Vol. COM-33, No. 11, November 1985, pp. 1191-1199.

37. Kleinrock, L., "Distributed Systems," invited paper for *ACM/IEEE-CS* Joint Special Issue (November 1985): *Communications of the ACM*, Vol. 28, No. 11, pp. 1200-1213, and *Computer*, Vol. 18, No. 11, pp. 90-103.

38. Rodrigues, P., Fratta, L., and M. Gerla, "Tokenless Protocols for Fiber Optic Local Area Networks," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-3, No. 6, November 1985, pp. 928-940.

39. "Improvements in the Time Complexity of Two Message-Optimal Election Algorithms", *Proceedings of the Fourth Annual ACM Symposium on Distributed Computing*, Miniaki, Canada, pp. 175-185, August 1985.

40. "Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks", *Proceedings of the Fourth Annual ACM Symposium on Distributed Computing*, Miniaki, Canada, pp. 186-195, August 1985.
Also appeared as "Simple and Efficient Distributed Algorithms for Election in Complete Networks", In the *Proceedings of the 22nd Annual Allerton Conference on Communication Control and Computing*, Urbana-Champaign, Ill., October 1984. (with Y. Afek).

41. Dechter, R. and L. Kleinrock, "Broadcast Communications and Distributed Algorithms," *IEEE Transactions on Computers*, Vol. C-35, No. 3, March 1986, pp. 210-219.

42. Levy, H. and L. Kleinrock, "A Queue with Starter and a Queue with Vacations: Delay Analysis by Decomposition," *Operations Research*, Vol. 34, No. 3, May-June 1986 (cover article), pp. 426-436.

43. Takagi, H. and L. Kleinrock, "Approximate Output Processes in Hidden-User Packet Radio Systems," *IEEE Transactions on Communications*, Vol. COM-34, No. 7, July 1986, pp. 685-693.

44. Marsan, M. A., and M. Gerla, "TOKENET -- A High Speed Token Reservation LAN," *Proceedings of ICCC 1986 (The 8th International Conference on Computer Communication)*, Munich, September 1986, pp. 259-264.

45. Gerla, M., "Routing and Flow Control in ISDN's," *Proceedings of ICCC 1986 (The 8th International Conference on Computer Communication)*, Munich, September 1986, pp. 643-647.

46. Belghith, A. and L. Kleinrock, "Average Response Time in Parallel Processing Systems with Certain Synchronization Constraints," UCLA, Computer Science Department Report No. 860033, November 1986.

47. Kleinrock, L., "Performance Models for Distributed Systems," invited paper (and keynote presentation) for *Teletraffic Analysis and Computer Performance Evaluation*, the Proceedings of the International Seminar held at The Centre for Mathematics and Computer Science June 2-6, 1986, Amsterdam; Elsevier Science Pub. B.V., Amsterdam, 1986, pp. 1-16.

48. Kleinrock, L. and J. Silvester, "Spatial Reuse in Multihop Packet Radio Networks," invited paper for *Proceedings of the IEEE*, Special Issue on Packet Radio Networks, Vol. 75, No. 1, January 1987, pp. 156-167.

49. Takagi, H. and L. Kleinrock, "Correction to 'Throughput Analysis for Persistent CSMA Systems'," Correspondence item, *IEEE Transactions on Communications*, Vol. COM-35, No. 2, February 1987, pp. 243-245.

50. Kleinrock, L. and H. Levy, "On the Behavior of a Very Fast Bidirectional Bus Network," Proceedings of the *IEEE International Conference on Communications (ICC '87)*, June 1987, pp 1419-1426

51. Felderman, R E., "Extension to the Rude CSMA Analysis," Correspondence item, *IEEE Transactions on Communications*, Vol. COM-35, No. 8, August 1987, pp. 848-849

PAPERS ACCEPTED FOR PUBLICATION

1. Levy, H. and L. Kleinrock, "The Analysis of Random Polling Systems," to be published in *Operations Research*, 1987.

PAPERS SUBMITTED FOR PUBLICATION

1. Akavia, G. and L. Kleinrock, "Hierarchical Use of Dedicated Channels," submitted to *Performance Evaluation*, June 1987.

2. Schooler, E., R. Felderman, and L. Kleinrock, "The Benevolent Bandit Laboratory: A Testbed for Distributed Algorithms", submitted for publication in *IEEE Journal on Selected Areas in Communications, Communications for Personal Computers II*, November 1987.

PAPERS IN PREPARATION

1. Gerla, M. and L. Kleinrock, "Congestion Control in Interconnected LAN's", *IEEE Network, October 1987.*

2. Kleinrock L. and H. Levy, "On The Behavior Of A Very Fast Bidirectional Bus Network", Reprinted from IEEE Communications Society, *IEEE International Conference on Communications*, June 7-10, 1987.

3. Levy, H. and L. Kleinrock, "Polling Systems With Zero Switch-Over Periods: A General Method for Analyzing the Expected Delay", to be submitted for publication in the *IEEE Transactions, '87 - '88.*

DISTRIBUTED SYSTEMS

Growth of distributed systems has attained unstoppable momentum. If we better understood how to think about, analyze, and design distributed systems, we could direct their implementation with more confidence.

LEONARD KLEINROCK

DISTRIBUTED SYSTEMS IN NATURE

How did the killer bees find their way up to North America? By what mechanism does a colony of ants carry out its complex tasks? What guides and controls a flock of birds or a school of fish? The answers to these questions involve examples of loosely coupled systems that achieve a common goal with distributed control.

Throughout nature we find an enormous amount of processing taking place at the level of the individual organism (be it an ant, a sparrow, or a human), and we have only begun to comprehend how processing and memory functions operate, especially in the human species. How does a human perform the acts of perception, cognition, decision making, and motor control? This processing occurs in a fraction of a second, using natural processing elements that are orders of magnitude slower than our current computer processing elements [8].

We do know that the brain is organized and structured very differently from our present computing machines. In human beings (i.e., in their internal neural systems) and in groups of organisms, nature has been extremely successful in implementing distributed systems that are far more clever and impressive than any computing machine humans have yet devised. We have succeeded in manufacturing highly complex devices capable of high-speed computation and massive accurate memory, but we have not yet gained sufficient understanding of distributed systems—our systems are still highly constrained and rigid in their construction and behavior. The gap between natural and man-made systems is huge, and we have a long way to go before we bridge the gap in understanding and implementation (see Figure 1, pp. 1202–1203).

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-82-C0064.

WHY SHOULD WE STUDY DISTRIBUTED SYSTEMS?

Currently we are experiencing the effects of the confluence of powerful forces in information technology. By far, the most significant effect is the host of revolutionary changes that have been brought about by the integrated chip—especially in the form of VLSI and the resulting enormous improvements in processing, storage, and communications. At the same time, we are experiencing a frightening backlog in software-application development while the user community is clamoring for unprecedented power in processing, communications, storage, and applications. Fortunately, we have the potential for this power—if only we could figure out how to put all the pieces together!

Distributed systems have come into existence in our industrial society in some very natural ways. For example, we have seen the emergence of a large number of distributed databases—systems that have evolved because the source of the data is not centralized and where there is a local need for frequent and immediate access to the locally generated data (e.g., the employee database at a branch office of a nationwide organization) in addition to a global need to view the entire database. Situations such as these require us to place some processing power at the many distributed locations for collecting, preprocessing, and accessing data. On-line transaction processing is an application that may contain a local component as well as a distributed-processing component, and the current proliferation of desktop personal computers is a manifestation of distributed-processing power. Indeed, if we measure

processing power in MIPS (millions of instructions per second), we note that the number of installed MIPS in personal computers is an order of magnitude greater than the number installed in mainframes. However, most of those PC MIPS lie idle most of the time. Imagine what a terrific distributed-processing system we could fire up with that unused power! When data and processing are distributed, we are obliged to provide communications to link the resources. Thus we are led into the use of packet networks, satellite networks, internets, cellular and packet radio networks, metropolitan area networks, and local area networks.

Distributed systems can provide the necessary power to meet the growing demands of the user community. We are demanding capability faster than the advances in devices alone can supply, and to meet these demands we will have to rely on innovative computing architectures such as parallel-processing systems. These large distributed databases, along with distributed-processing and distributed-communication networks, have given rise to some very complex distributed-system structures, and it is essential that we learn how to think about them properly (see Figure 2, pp. 1204–1205).

ARCHITECTURE AND ALGORITHMS

The world of applications has an insatiable need for computing power. A good mathematician can easily consume any finite computing capability by posing a combinatoric problem whose computational complexity grows exponentially with a variable of the problem (e.g., the enumeration of all graphs with N nodes). The ways in which we push back this "power wall" involve both hardware and software solutions. Typically, the methods for speeding up the computation include the following:

- faster devices (a physics and engineering problem),
- architectures that permit concurrent processing (a system design problem),
- optimizing compilers for detecting concurrency (a software-engineering problem),
- algorithms for specification of concurrency (a language problem), and
- more expressive models of computation (an analytic problem).

Characterizing the Architecture

There are many ways of classifying machine architectures—too many, in fact. The following classification was selected for the purposes of this article.

We begin with the purely serial uniprocessor in which a single instruction stream operates on a single data stream (SISD). These systems are "centralized" at the global level, but really do contain many elements of a distributed system at the lower levels, for example, at the level of communications on the VLSI chips themselves.

Next is the vector machine, in which a single instruction stream operates on a multiple data stream (SIMD). These include array processors (e.g., systolic arrays) and pipeline processors.

The third consists of multiple processors that, collectively, can process multiple instruction streams on multiple data streams (MIMD). The form of multiprocessing that takes place when multiple processors cooperate closely to process tasks from the same job is referred to as parallel processing. On the other hand, the term distributed processing is applied to the form of multiprocessing that takes place when the multiple processors cooperate loosely and process separate jobs.

Vector machines and multiprocessing systems all provide some form of concurrency. The effect of this concurrency on system performance is important and is therefore a very active area of research (see Figure 3, p. 1206).

Since the onslaught of the VLSI revolution, a number of machine architectures have been implemented in an attempt to provide the supercomputing power toward which concurrent processing tempts us [5]. Two excellent recent summaries of some of these projects are offered by Hwang [9] and by Schneck et al. [17]. There you will find the Butterfly machine, the Cosmic Cube, various kinds of tree machines, the Cedar project, the Sisal language, the Connection machine, and others whose names are intriguingly close to Mother Nature's systems.

Characterizing the Algorithm

The major goal in characterizing the algorithm is to identify and exploit its inherent parallelism (i.e., potential for concurrency). The levels of resolution at which we can attempt to find this parallelism are listed below in decreasing order of granularity [16]:

- job execution,
- task execution,
- process execution,
- instruction execution,
- register transfer, and
- logic device.

Clearly, as we drop down the list to finer granularity, we expose more and more parallelism, but we also increase the complexity of scheduling these tiny objects to the processors and of providing the communications among so many objects (the problem of interprocess communication—IPC). As was stated earlier, if we operate at the top level (i.e., at the job level), then we think of the system as a distributed-processing system; if we operate at the task or process level, we have a parallel-processing system; if we operate at the instruction level, we have the vector machine and the array processor.

Regardless of the level at which we operate, it behooves us to create a "model" of the algorithm or, if you will, of the computation we are processing [10]. A very common model is the graph model of computation, which is normally used at the task or process level

(another common modeling method is the use of Petri Nets). In this model, the nodes represent the tasks (or processes), and the directed edges represent the dependencies among the tasks, thereby displaying the partial ordering of the tasks and the parallelism that can be exploited (see Figure 4, p. 1207).

However, the problem of finding the parallelism in the lines of code that represent the algorithm still remains, and there is an ongoing effort to simplify (and even automate) this task by developing parallel programming languages for implementing these algorithms (e.g., Ada®, concurrent Pascal).

Matching the Architecture to the Algorithm

The performance of a distributed system depends strongly on how well the architecture and the algorithm are matched. For example, a highly parallel algorithm will perform well on a highly parallel architecture; a distributed system requiring lots of interprocessor communication will perform poorly if the communication bandwidth is too narrow. This matching problem becomes fierce and crucial when we attempt to coordinate an exponentially growing number of processors requiring an exponentially growing amount of interprocessor communication. The apparent solution to such an unmanageable problem is one that is self-organizing.

If we choose to use the graph model discussed, we are faced with a number of architecture/algorithm problems, namely, partitioning, scheduling, memory access, interprocess communication, and synchronization. The partitioning problem refers to decisions regarding the level of granularity and the choices involving which objects should be grouped into the same node of the task graph. The scheduling problem refers to the assignment of processors and memory modules to nodes of the computation graph. In general, this is an NP-complete problem (tough as nails to do optimally). The memory-access problem refers to the mechanism that allows processors to communicate with the various memory modules; usually, either shared-memory or message-passing schemes are used. The interprocessor-communication problem refers to the nature of the communication paths and connections that are available to provide processors access to the memory modules and to other processors; this may take the form of an interconnection network in a parallel-processing system, a local area network in a local distributed-processing system or shared data system or shared peripheral system, or a packet-switched, value-added, long-haul network in a nationwide distributed system. Synchronization refers to the requirement that no node in the graph model can begin execution until all of its predecessor nodes have completed their execution.

The use of broadcast or multicast communication opens up a number of interesting alternatives for communication. Local area networks take exquisite advantage of these communication modes. Algorithms that require tight coupling (i.e., lots of IPC) need not only large bandwidths (which, for example, could be provided by fiber-optic channels), but also low latency. Specifically, the speed of light introduces a 15,000-microsecond latency delay for a communication that must travel from coast-to-coast across the United States.

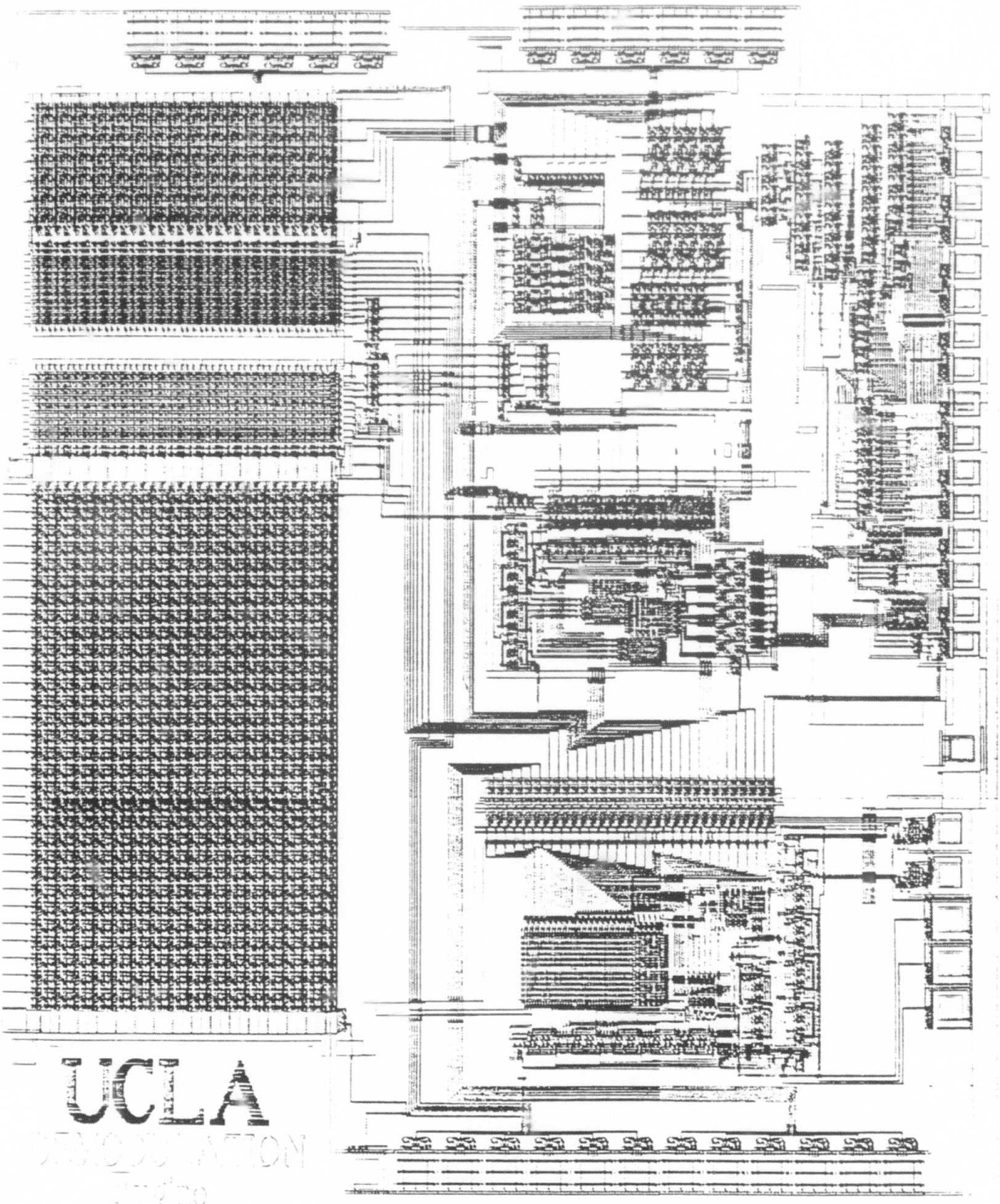
Another consideration in matching architectures to algorithms is the balance and trade-off among communication, processing, and storage. We have all seen sys-

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

(a)

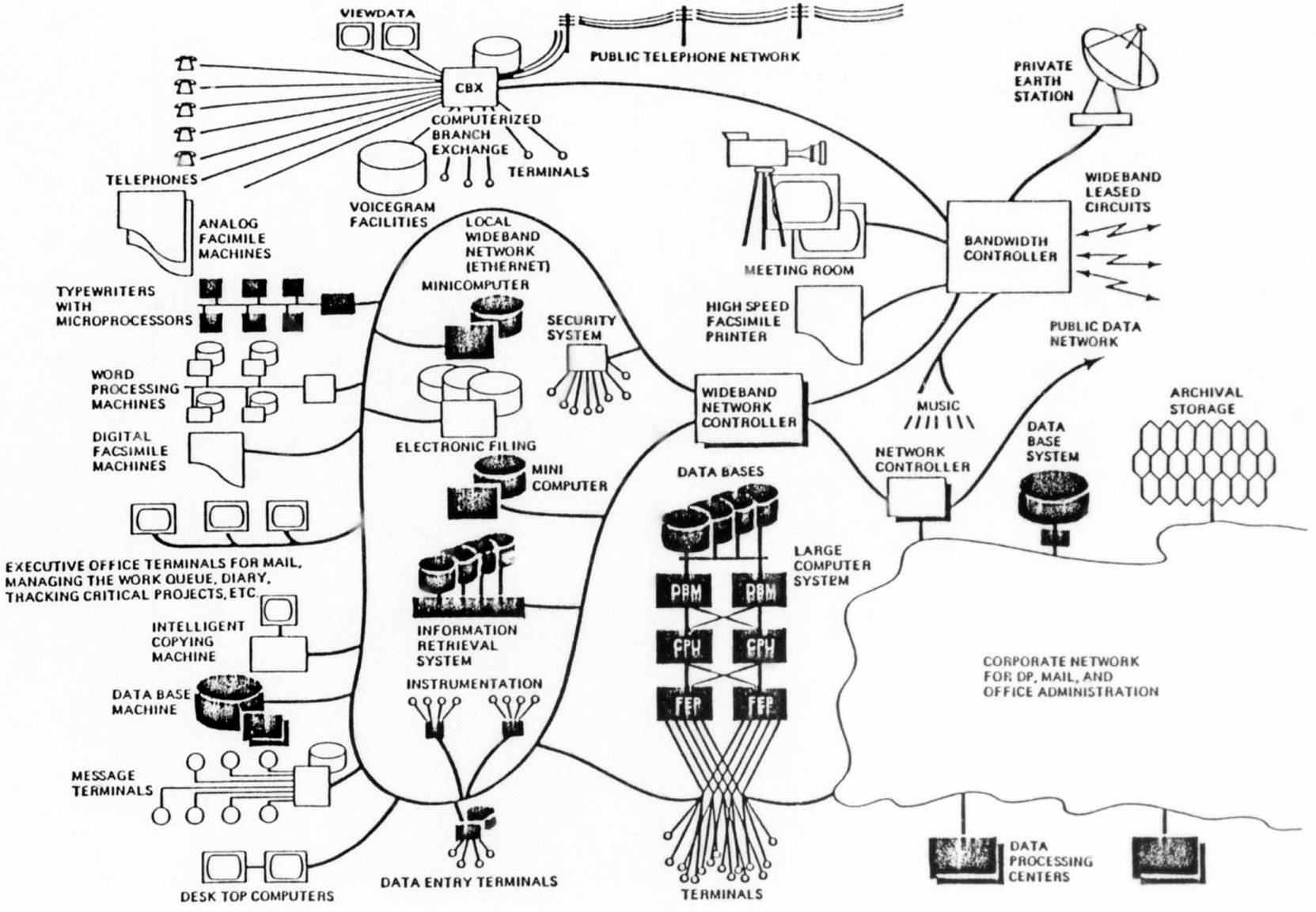
There is an amazing contrast between the neural structure of the human brain (a) and the architecture of today's VLSI chips (b). The brain is massively parallel, densely (and weirdly) connected with leaky transmission paths, highly fault tolerant, self-repairing, adaptive, noisy, and probably nondeterministic. Man-made computers are highly constrained, precisely (and often symmetrically) laid out with carefully isolated wires, not very fault tolerant, largely serial and centralized, deterministic, minimally adaptive, and hardly self-repairing. (Photo (a) is the courtesy of Peter Arnold, Inc.)

FIGURE 1. Natural and Man-Made Architectures



(b)

FIGURE 1. Natural and Man-Made Architectures


VIEWDATA
PUBLIC TELEPHONE NETWORK
PRIVATE EARTH STATION
CBX
COMPUTERIZED BRANCH EXCHANGE
TERMINALS
TELEPHONES
VOICEGRAM FACILITIES
ANALOG FACIMILE MACHINES
LOCAL WIDEBAND NETWORK (ETHERNET)
MEETING ROOM
WIDEBAND LEASED CIRCUITS
BANDWIDTH CONTROLLER
TYPEWRITERS WITH MICROPROCESSORS
MINICOMPUTER
SECURITY SYSTEM
HIGH SPEED FACSIMILE PRINTER
PUBLIC DATA NETWORK
WORD PROCESSING MACHINES
ELECTRONIC FILING
WIDEBAND NETWORK CONTROLLER
MUSIC
ARCHIVAL STORAGE
DIGITAL FACSIMILE MACHINES
MINI COMPUTER
DATA BASES
NETWORK CONTROLLER
DATA BASE SYSTEM
EXECUTIVE OFFICE TERMINALS FOR MAIL, MANAGING THE WORK QUEUE, DIARY, TRACKING CRITICAL PROJECTS, ETC.
LARGE COMPUTER SYSTEM
PBM
PBM
CPU
CPU
FEP
FEP
INTELLIGENT COPYING MACHINE
INFORMATION RETRIEVAL SYSTEM
CORPORATE NETWORK FOR DP, MAIL, AND OFFICE ADMINISTRATION
DATA BASE MACHINE
INSTRUMENTATION
MESSAGE TERMINALS
DATA PROCESSING CENTERS
DESK TOP COMPUTERS
DATA ENTRY TERMINALS
TERMINALS

tems where one of these resources can be exchanged for others. For example, if we do some preprocessing in the form of data compression prior to transmission, we can cut down on the communication load (trade processing for communication). If we store a list of computational results, we can cut down on the need to recompute the elements of the list each time we need the same entry (trade storage for processing). Similarly, if we store data from a previous communication, we need merely transmit the data address or name of the previous message rather than the message itself (trade storage for communication). Selecting the appropriate mix in a given problem setting is an important issue.

Distributed algorithms operating in a distributed network environment (e.g., a packet-switched network) pose the possibility that network failures may cause the network to temporarily be partitioned into two (or more) isolated subnetworks. In such a case, detection and recovery mechanisms must be introduced (see Figure 5, p. 1207).

Lastly, it should be mentioned that very little is known about characterizing those properties of an algorithm that cause it to perform well or poorly in a distributed environment.

PERFORMANCE AND BEHAVIOR

We do know some things about the way distributed systems behave, precious few though they may be. The most interesting thing about them is that they come to us from research in very different fields of study. Unfortunately, the collection of results (of which the following is a sample) is just that—a collection, with no fundamental models or theory behind it.

We begin by considering closely coupled systems, in particular, parallel-processing systems. One of the most compelling applications of parallel processing is in the area of scientific computing, where the speed of the world's largest uniprocessors is hopelessly inadequate to handle the computational complexity required for many of these problems [3]. Of course, the idea is that, as we apply more parallel processors to the computational job, the time to complete that job will drop in proportion to the number of (identical) processors, P. The "speedup" factor, denoted by S, is a common measure of performance for parallel-processing systems and is defined as the time required to complete the job using P processors, divided into the time required to complete the job using one of these processors. S may also be interpreted as the average number of busy processors, that is, the concurrency. The best we can achieve is for S to grow directly with P; that is,

$$S \leq P.$$

Thus, in general, $1 \leq S \leq P$. In the early days of parallel processing, Minsky [15] conjectured a depressingly pessimistic form for the typical speedup; namely,

$$S = \log P.$$

Often that kind of poor performance is indeed observed. Fortunately, however, experience has shown that things need not be that bad. For example, we can achieve $S = 0.3P$ for certain programs by carefully extracting the parallelism in Fortran DO loops [14]. However, Amdahl has pointed out a serious limitation to the practical improvements one can achieve with parallel processing (the same argument applies to the improvements available with vector machines) [1]. He argues that, if a fraction, f, of a computation must be done serially, then the fastest that S can grow with P is

$$S_{\max} = \frac{P}{fP + 1 - f}.$$

We see that, for $f = 1$ (everything must be serial), $S_{\max} = 1$; for $f = 0$ (everything in parallel), $S_{\max} = P$.

The actual amount of parallelism (i.e., S) achieved in a parallel-processing system is a quantity that we would like to be able to compute. S is a strong function of the structure of the computational graph of the jobs being processed. I, with one of my students [2], have been able to calculate S exactly as a simple function of the graph model. Specifically, we consider a parallel-processing system with P processors and with an arrival rate of λ jobs per second. We assume the collection of jobs can be modeled with an arbitrary computation graph with an average of N tasks per job, each task requiring an average of $\bar{x}$ seconds. Then it can be shown that

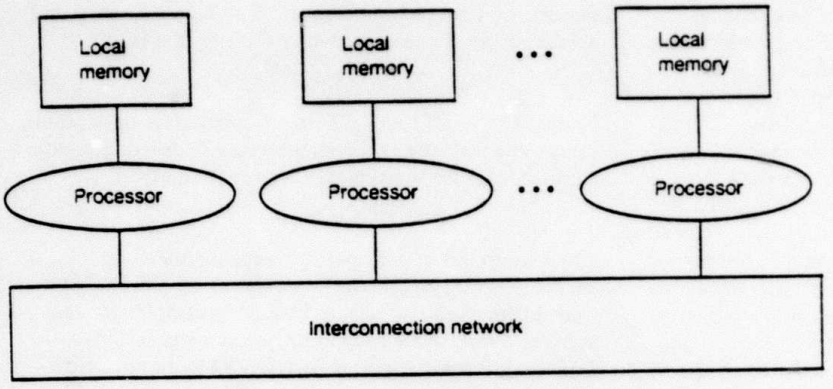
$$S = \begin{cases} \lambda N \bar{x} & \text{for } \lambda N \bar{x} \leq P \\ P & \text{for } \lambda N \bar{x} \geq P. \end{cases}$$

This is a very general result; in some special cases, the distribution of the number of busy processors can be found as well.

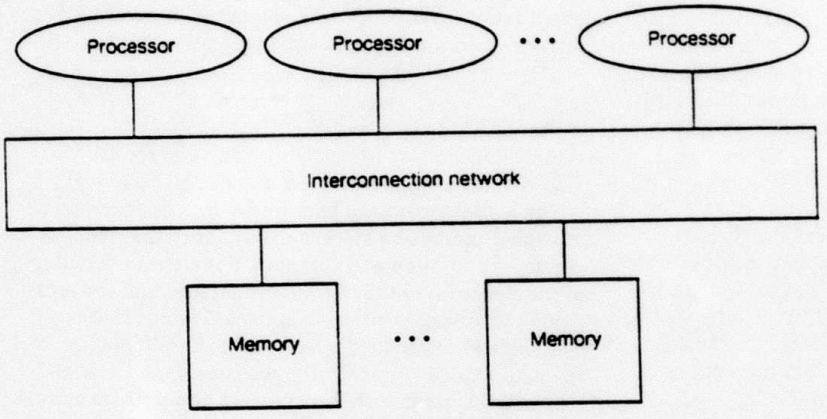
So far, we have given ourselves the luxury of increasing the system's computational capacity as we have added more processors to the system. Let us now consider adding more processors, but in a fashion that maintains a constant total system capacity (i.e., a constant system throughput in jobs completed per second). This will allow us to see the effect of *distributing* the computation for a job over many smaller processors. The particular structure we are considering is the regular series-parallel structure shown in Figure 6 (p. 1208), where we have taken a total processing capacity of C MIPS and divided it equally into mn processors, each of C/mn MIPS. On entering the system, a job selects

FIGURE 2. A Complex Distributed System (left)

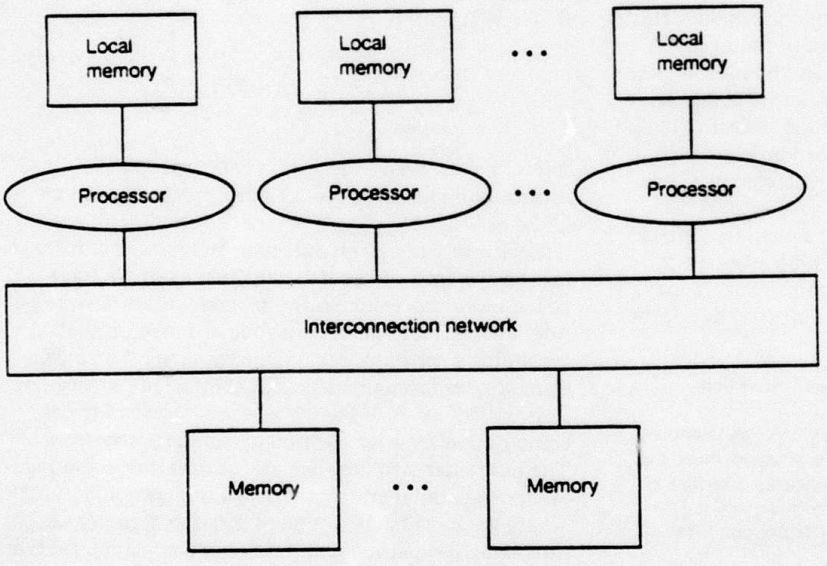
Humans have created some unbelievably complex distributed systems. The fact that they work at all is amazing, given that we have not yet uncovered the basic principles determining their behavior. (From Martin, J. *Design and Strategy for Distributed Data Processing*. Prentice-Hall, Englewood Cliffs, N.J., 1981.)



(a) **Message-Passing Architecture: Local Memory**



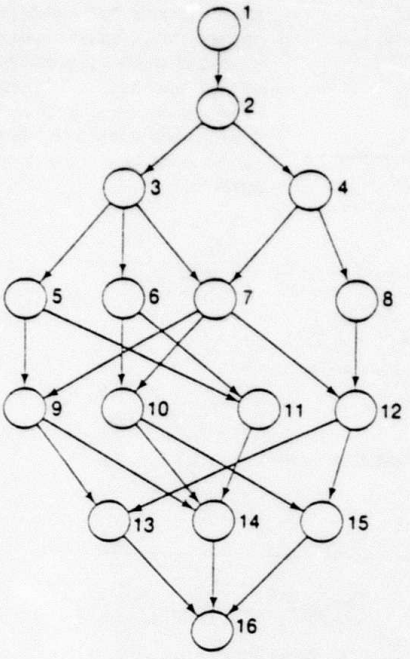
(b) **Shared-Memory Architecture**



(c) **Hybrid Architecture**

FIGURE 3. Architectures for Connecting Processors and Memory

Locality of memory reference, bandwidth of communication, processor overhead, and cost are key issues determining the appropriate architecture for a given application.



The graph model of computation is an extremely useful model for displaying the parallelism inherent in an algorithm (i.e., a job). The entire graph represents the computational tasks associated with that job, the nodes represent the tasks themselves, and the directed arcs, which define a partial ordering of the nodes, represent the sequence in which the tasks must be performed.

FIGURE 4. Graph Model of Computation

(equally likely) any one of the m series branches down which it will travel. It will receive $1/n$ of its total processing needs at each of the n series-connected processors. The key result for this system [13] is that the mean response time for jobs in this series-parallel pipeline system is mn times as large as it would have been had the jobs been processed by a single processor of C MIPS! There are some statistical assumptions behind this result, but the message is clear—distributed processing of this kind is terrible. Why, then, is everyone talking about the advantages of distributed processing? The answer must be that a large number of small processors (e.g., microprocessors) with an aggregate capacity of C MIPS is less expensive than a large uniprocessor of the same total capacity. It can be shown that the series-parallel system *will* have the same response time as the uniprocessor if the aggregate capacity of the series-parallel system has K times the capacity of the uniprocessor where

$$K = mn - \rho(mn - 1)$$

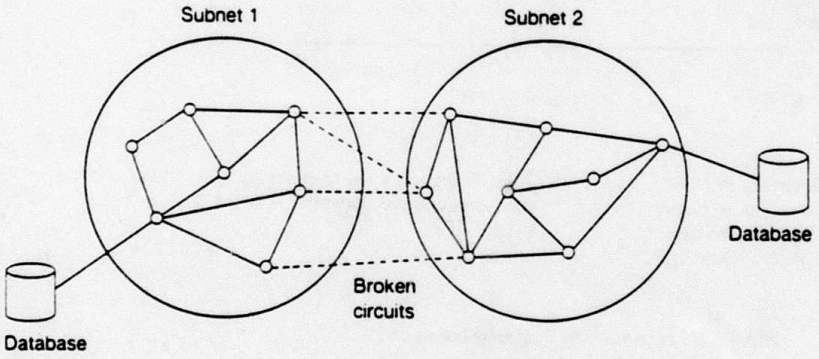
and where ρ is the utilization factor for each processor; namely, ρ = arrival rate of jobs times the average service time per job for a processor. This says that, for light loads ($\rho \ll 1$), $K = mn$, whereas, for heavy loads ($\rho \to 1$), $K = 1$. Is it the case that smaller machines are mn times less expensive than larger machines (so that we can purchase mn times the capacity at the same total price, as is needed in the light-load case)? To answer this question, recall a law that was empirically observed by Grosch more than three decades ago. Grosch's law [7] states that the capacity of a computer is related to its cost, which we denote by D (dollars) through the following equation:

$$C = jD^2$$

where j is a constant. This law may be rewritten as

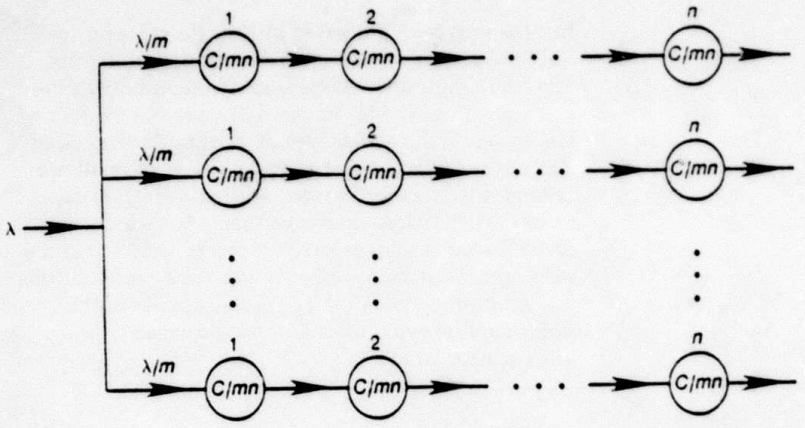
$$\frac{D}{C} = \frac{1}{\sqrt{jC}}.$$

Grosch tells us that the economics are *exactly the reverse* of what we need to break even with distributed processing! He says that larger machines are cheaper per MIPS. If Grosch is correct today, then why are microprocessors selling like hotcakes? A more recent look at the economics explains why. Ein-Dor [4] shows that, if we consider all computers at the same time, Grosch's law is clearly not true, as seen in Figure 7 (p. 1208). In this figure we see that microcomputers are a good buy.



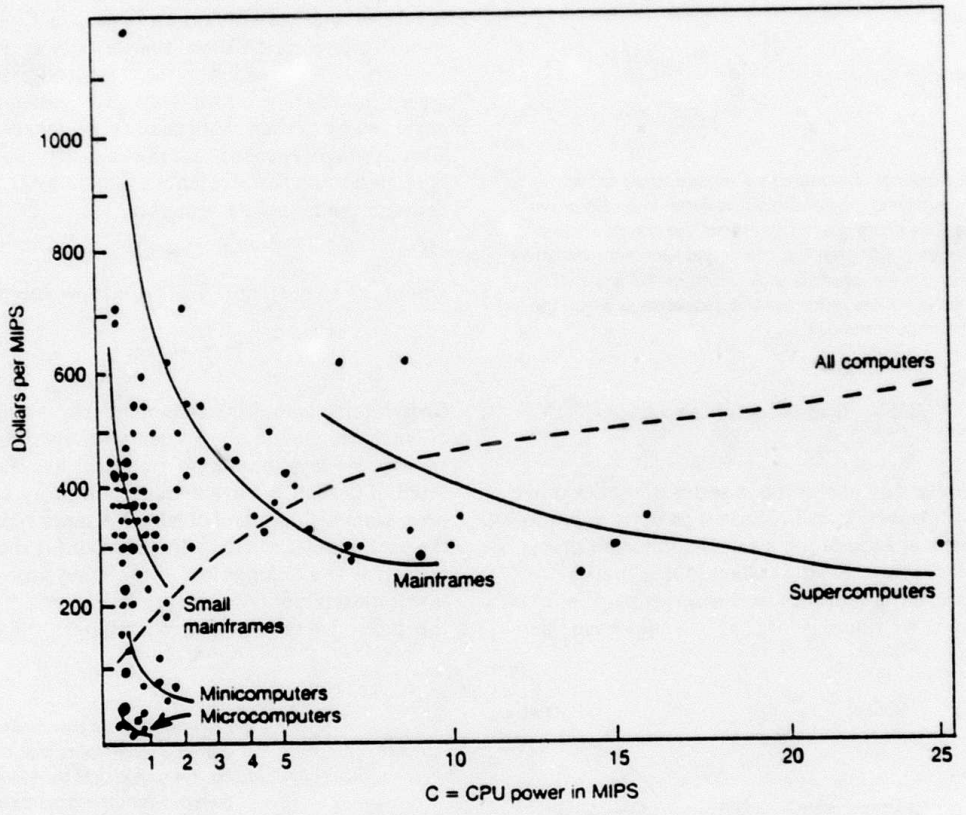
Network failures can create two separated subnetworks that cannot communicate until the failure is repaired. Maintaining consistency of databases in such a situation is a key issue in distributed-systems design.

FIGURE 5. A Partitioned Network



When a constant amount of processing capacity (C) is distributed into *mn* equal (and smaller) processors in a network such as this, the response time increases by a factor of *mn*. How can one justify a distributed system in the face of this degradation?

FIGURE 6. A Symmetrical Distributed-Processing Network



The cost per MIPS seems to rise with the number of MIPS when we examine all computers in a single group. However, when we separate them into families, we find that the opposite is true, thus confirming Grosch's law. Figure taken from Ein-Dor, P. Grosch's law re-revisited: CPU power and the cost of computation. *Commun. ACM 28*, 2 (Feb. 1985), 142–151.

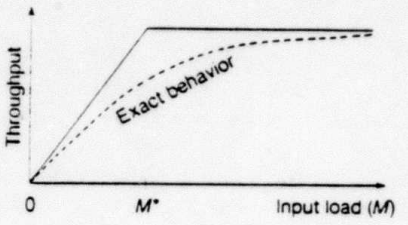
FIGURE 7. Economics of Computer Power

However, as Ein-Dor points out, Grosch's law is still true today if we consider *families* of computers. Each family has a decreasing cost per unit of capacity as capacity is increased. Ein-Dor goes on to make the observation that, if one needs a certain number of MIPS, then one should purchase computers from the smallest family that can currently supply that many MIPS. Furthermore, once in the family, it pays to purchase the biggest member machine in that family (as predicted by Grosch).

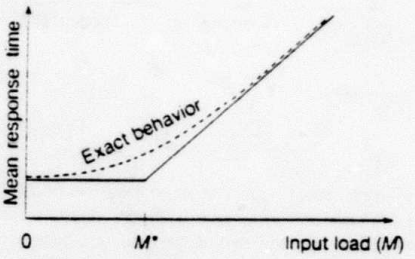
Now that we have discussed the performance of parallel-processing systems for some special cases, let us generalize the ways in which jobs pass through a multiprocessor system, and analyze the system throughput and response time. Indeed, we bound these key system-performance measures in the following way: Suppose we have a population of M customers competing for the resources of the system. Assume that customers generate jobs to be processed by some of the system's resources, that the way in which these jobs bounce around among the resources is specified in a probabilistic fashion, and that the mean response time of this system is T seconds. When a customer's job leaves the system, that customer then begins to generate another job request for the system, where the average time to generate the request is t_o seconds. Of interest is the mean response time, T, and the system throughput γ as a function of the other system parameters. Although we have been extremely general in the system description, we can nevertheless place an excellent upper bound on the system throughput and an excellent lower bound on the mean response time as shown in Figure 8. In this figure, the quantity M^* is defined as the ratio of the mean cycle time $T_o + t_o$ to the mean time x_o required on the critical resource in a cycle: T_o is the mean response time when $M = 1$, and the critical resource is that system resource that is most heavily loaded [11].

To find the exact behavior (shown in dashed lines in the figure) rather than the bounds, one must be much more explicit about the distributions of the service time required by jobs at each resource in the system as well as the queueing discipline at each. Using the bounds or the exact results, the effect of parameter changes on the system behavior can be seen. For example, one can examine the accuracy of the common rule of thumb that suggests that the proper mix of microprocessor speed, memory size, and communication bandwidth is in the proportion 1 MIPS, 1 Mbyte, and 1 Mbit per second; some suggest that we will soon see a 10, 10, 10 mix instead of the 1, 1, 1 mix. Of course the correct answer to this question depends on the total system configuration.

Once we evaluate the throughput and mean response time for a system, we usually want to find the relationship between the two, which typically has the well-known shape (shown in Figure 9, p. 1210) that clearly demonstrates the trade-off between them—a low delay implies a small throughput and vice versa.



(a) Bound on Throughput



(b) Bound on Mean Response Time

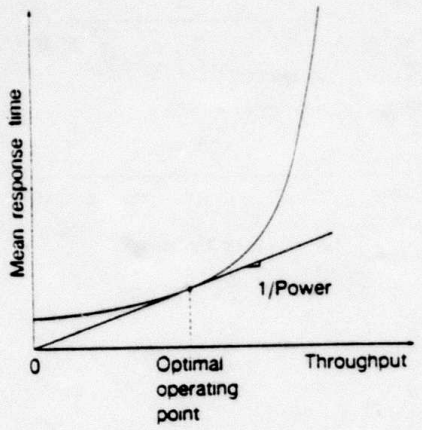
Excellent bounds on throughput (a) and mean response time (b) as a function of the number of users (or any measure of the input load) are easily obtained for a very large class of distributed systems. The exact behavior can be derived for more restricted systems and demonstrates the excellence of the bounds.

FIGURE 8. Bounds on Throughput and Response Time

We are immediately compelled to inquire about the location of the "optimal" operating point for a system. The answer depends on how much you hate delay versus how much you love throughput. One way to quantify this love–hate choice is to define a quantity known as "power" (denoted by P), which is defined as

$$P = \frac{\gamma}{T}.$$

The operating point that optimizes (i.e., maximizes) the power (large throughput and small delay) is located at that throughput where a straight line (of minimum slope) out of the origin touches the throughput-delay profile (usually tangentially); such a tangent and operating point are shown in Figure 9. This result holds for all profiles and all flow-control functions (see below). Moreover, for a large class of queueing curves, this optimal operating point implies that the system should be loaded in such a way that each resource has, on the *average, exactly one job* to work on [12].



The delay–throughput relationship, an example of the key profile in systems performance evaluation, clearly shows the trade-off between the two. In general, you cannot get a small delay and a large throughput at the same time. We can, however, maximize "power," which is the ratio of throughput to delay, in order to define the natural point for a system.

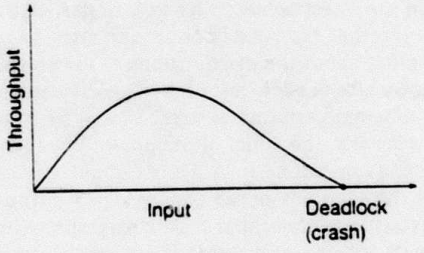
FIGURE 9. The Key System Profile

Unfortunately, there are some distributed systems that do not have the nice relationship shown in Figure 8a where the throughput rises asymptotically to its maximum value as the "input" is increased. Often we find the behavior depicted in Figure 10 where the throughput reaches a peak and then declines as the input increases further, possibly dropping to zero, in which case we say that the system has crashed. Such behavior has been observed in paged virtual-memory systems (thrashing), in computer networks (deadlocks and degradations), and in automobile traffic flow (bumper-to-bumper traffic). Here again, one must find a method for controlling the input (i.e., setting the system operating point) so as to achieve optimal or near-optimal performance (somewhere near the peak of the curve in Figure 10).

"Flow control" is the name associated with this operation, and it can be implemented in a centralized or a distributed fashion in distributed systems with the latter being the more challenging design problem [6]. One example of distributed control is the dynamic routing procedure found in many of today's packet-switching networks where no single switching node is responsible for the network routing. Instead, all nodes participate in the selection of network routes in a distributed fashion. A great deal of research is currently under way to evaluate the performance of other distributed algorithms in networks and distributed systems. Examples are the distributed election of a leader, distributed rules for traversing all the links of a network, and distributed rules for controlling access to a database.

Another large class of distributed-control algorithms has to do with sharing a common communication channel among a number of devices in a distributed fashion [19]. If the channel is a broadcast channel (also known as a one-hop channel), then the analytic and design problem is fairly manageable and a number of popular local area network algorithms for media access control have been studied and implemented. Examples here include CSMA/CD (carrier-sense multiple access with collision detect—as used in Xerox's Ethernet, AT&T's 3B-Net and Starlan, and IBM's PC Network), token passing (as used in the token-ring and token-bus networks), and address contention resolution (as used in AT&T's ISN). A large number of additional channel access algorithms have been studied in the literature including Expressnet, tree algorithms, urn models, and hybrid models. If the channel is multicast (or multihop), then the analytic problem becomes much harder.

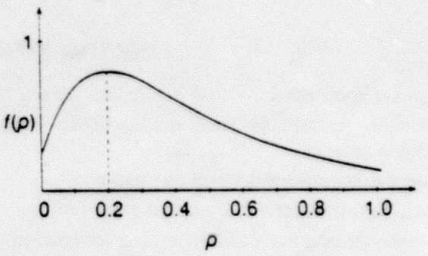
But what if the processors in our distributed environment are allowed to communicate with their peers in very limited ways? Can we endow these processors (let us call them automatons for this discussion) with an internal algorithm that will allow them to achieve a collective goal? Tsetlin [20] studied this problem at length and was able to demonstrate some remarkable behavior. For example, he describes the Goore game in which the automatons possess finite memory and act in a probabilistic fashion based on their current state and the current input. They cannot communicate with each other at all and are required to vote YES or NO at



There are many systems that degrade badly when pushed too hard. They can even degrade to a situation of deadlock. Examples include thrashing in virtual memory systems, deadlocks in computer networks, and bumper-to-bumper traffic in highway systems.

FIGURE 10. A Dangerous Throughput Profile

certain times. The automatons are not aware of each other's vote; however, there is a referee who can observe and calculate the percentage, p, of automatons that vote YES. The referee has a function, $f(p)$ (such as that shown in Figure 11), where we require that $0 \leq f(p) \leq 1$. Whenever the referee observes a percentage, p, who vote YES, he or she will, with probability, $f(p)$, reward each automaton, independently, with a one dollar payment; with probability $1 - f(p)$ he or she will punish an automaton by taking one dollar away. Tsetlin proved that no matter how many players there may be in a Goore game, if the automatons have sufficient memory, then for the payoff probability shown in the figure, exactly 20 percent of the automatons will vote YES with probability one! This is a beautiful demonstration of the ability of a distributed-processing system to act in an optimum fashion, even when the rules of the reward function are unknown to the players and when they can neither observe nor communicate with each other. All they are allowed is to vote when asked, and to observe the reward or penalty they receive as a result of that vote. In this work we see the beginnings of a theory that may be able to explain how the colony of ants performs its tasks.



The Goore game rewards each member of a set of automatons independently with a probability given by the function $f(p)$, where p is the fraction of the set that votes YES at a given time. The automatons are completely unaware of the other automatons, do not know the function $f(p)$, and, remarkably, will collectively vote in a way that maximizes the payoff to all.

FIGURE 11. The Gocre Game

NEEDED UNDERSTANDING AND TOOLS

In the previous section, we discussed a few of the things known about distributed-systems performance and behavior. A few isolated facts are indeed known, but overall theory and understanding are still lacking.

For instance we need considerably sharper tools to evaluate the ways in which randomness, noise, and inaccurate measurements affect the performance of distributed systems. What is the effect of distributed control in an environment where that control is delayed, based on estimates, and not necessarily consistent throughout the system? What is the effect on performance of scaling some of the system parameters? We need a common metric for discussing the various system resources of communications, storage, and processing. For example, is there a processing component to communications? We also need a proper way to discuss distributed algorithms and distributed architectures.

A microscopic theory that deals with the interaction of each job with each component of the system is likely to overwhelm us with detail and will fail to lead us to an understanding of the overall system behavior. It is similar to the futility of studying the many-body problem in physics in order to obtain the global behavior of solids. What is needed is a macroscopic theory of distributed systems, such as thermodynamics has provided for the physicist. In fact, Yemini [21] has proposed an approach for a macroscopic theory based on statistical mechanics that will lead to better understanding the global behavior of distributed systems without the need for a detailed, fine-grained analysis.

Another fruitful approach that also avoids the horrible details of any specific system structure must be credited to Shannon [18]. In analyzing the behavior of error-correcting codes for noisy communication channels, Shannon used the brilliant device of studying *all possible codes simultaneously*. This enabled him to average out the detailed structure of any given code. He could then take exquisite advantage of the law of large numbers in order to arrive at a precise statement regarding the error behavior of codes. It is likely that such an approach will allow us to study the behavior of "typical" topologies and algorithms in distributed systems.

LIKELY FUTURE DEVELOPMENTS

These are exciting times. Researchers in universities and laboratories around the world have begun to focus their attention on distributed systems. They come to this field from diverse disciplines ranging from queueing theory to neuroanatomy in which they are the experts. Thus, we have the ingredients for an enormously rich soup of separate ideas that have only just begun to blend.

As the theoretical frontiers are being assaulted, so too are the practitioners busily building systems. This is a double-edged sword. On the one hand, the implementation of real distributed systems in the hands of the designers and users provides us with a strong motivation for progress in understanding, as well as a magnificent test bed in which we can experiment. On the other hand, these systems are massively expensive and are being implemented without the benefit of the principles we seek. As a result, they may be colossal failures! The reality is that there is no way we can prevent their proliferation as manufacturers respond to the frenzied demand from the user community. In a sense

UNDERLYING PRINCIPLES OF DISTRIBUTED-SYSTEMS BEHAVIOR

- Developing innovative architectures for parallel processing
- Providing better languages and algorithms for specification of concurrency
- More expressive models of computation
- Matching the architecture to the algorithm
- Understanding the trade-off among communication, processing, and storage
- Evaluation of the speedup factor for classes of algorithms and architectures
- Evaluation of the cost-effectiveness of distributed-processing networks
- Study of distributed algorithms in networks
- Investigation of how loosely coupled self-organizing automatons can demonstrate expedient behavior
- Development of a macroscopic theory of distributed systems
- Understanding how to average over algorithms, architectures, and topologies to provide meaningful measures of system performance

we are all responsible for the current craziness, because we have been "promising" these miraculous systems to the user for almost a decade.

In the face of these developments, we can foresee some of the likely developments that will take place over the next decade or two. Let us first consider the likely technology developments in hardware-type resources. One of the most exciting of these is the huge data bandwidth projected for fiber-optic technology. These fibers are being used for point-to-point communication pipes at rates on the order of hundreds of megabits per second. Bell Laboratories and Japan have been leapfrogging each other in setting world records for the largest data rates transmitted over the longest distances. Earlier this year, Bell Laboratories established a new record by transmitting at the rate of 4 billion bits per second at a distance of 117 km without any repeaters! The product of data rate times distance has been doubling every year since 1975, and based on the limits imposed by physics, there are still five orders of magnitude to go (16 years of doubling left). The tiny glass fiber is so clear that, if the oceans of the world were made of this glass, one could see the bottom of the deepest trench in the ocean floor from the surface. If we consider a 1-mW laser and a requirement of 10 photons to detect 1 bit of information (high-quality detection), then a single strand of fiber should be able to support a data bandwidth of 10^{15} bits per second. That would provide, for example, a 100-Mbit-per-second channel to each of 10 million users—all on one thin strand! This light-wave technology is being installed across the United States right now. The Los Angeles 1984 Olympics video was transmitted from the games' remote locations to satellite transmitters using a fiber-optic network installed by Pacific Bell—perhaps the most well-known application to date. This technology is being applied to local area networks by a number of vendors, but the technology for this application is not yet mature because we have yet to develop an efficient way to optically tap into the light pipes at low loss. As soon as that problem is resolved (in the next two or three years), we are likely to see a rapid deployment of fiber-optic channels in our local network environment. As discussed earlier, enormous bandwidths are necessary, but not sufficient, for many tightly coupled systems. The latency introduced due to propagation delay can inhibit tight control. (E.g., if we transmit data into a 1-Gbit-per-second light pipe spanning the United States, the 15,000-microsecond propagation delay is such that the first bit will come out of the other end only after 15 million bits have been pumped in!)

This planet is currently laced with many types of computer/communications networks at all levels. There are wide area networks, packet-switched networks, circuit-switched networks, satellite networks, packet radio networks, metropolitan area networks, local area networks, cellular radio networks, and more; and they are mostly incompatible within each type and across types. At the same time, the end user's facility consists of telephones, data terminals, Host machines, PBX switches, alarm systems, video systems, FAX machines, etc. The incompatibility problem escalates! What is needed in a distributed system is a standard digital communication service to connect the many user devices with one another across the room or across the world. Fortunately, there is a worldwide movement to define and adopt an integrated solution to this problem, which has given rise to the Integrated Services Digital Network (ISDN). The ISDN service defines a customer interface (a plug in the wall) to which the user's devices can attach and gain access to the worldwide integrated digital network. We are not likely to see much definition and penetration of ISDN until the end of this decade and, possibly, into the next decade (and most likely it will first appear at the local network level).

What all this should tell us is that we are approaching a time when massive connectivity among devices and systems will exist. Such connectivity is necessary if we are to derive the full benefits from distributed systems.

At the processor technology level, perhaps the most dramatic development is the gathering momentum in the proliferation of personal workstations. They are spearheading the drive toward distributed systems. At the other end of the spectrum, parallel machine archi-

tectures are being proposed all over the world to increase the processing capacity that can be applied to a single problem. Both of these technologies are moving very rapidly and are putting pressure on distributed-systems research and development. We are seeing the development of massively distributed architectures that can be configured as tightly coupled, loosely coupled, or even hierarchically structured systems.

Massively distributed and massively connected systems with enormous computational capacity are likely to appear in the next 10 years. Unless we pay very careful attention to the user interface, users will be hopelessly lost and ineffective. At the very least, we must provide users with languages that allow them to take advantage of the distributed architecture and to write application code quickly and in a way that allows the application package to be modified and maintained easily. Moreover, the complexity of the system should be transparent to users. Users need to interface with a systemwide operating system that offers the use of a single logon (with a networkwide name and password) and that provides access to file servers, database servers, automatic backup, processing servers, mail servers, application packages, education and help functions, etc.

The system itself could take advantage of expert-systems capability in providing these services to the user. And the system is likely to include extensive redundancy in order to provide high levels of reliability and fault tolerance. It should also be self-repairing, and even self-organizing, as the conditions and demands on it change.

Aside from the business-oriented applications and developments listed above, an enormous consumer-oriented set of products will be developed. One device that spans business and personal needs is a proper "lap" computer that will provide the user with remote access to the massive distributed network resources described in this article.

We foresee a new phenomenon whereby users are confronted with so many attractive features in new devices and software packages that they cannot possibly learn to use them all. Learning how to use the features represents an investment far beyond users' available time; and yet the features are wonderfully seductive. To coin a term, I would like to refer to this phenomenon as "FEATURE SHOCK"!

As we observe the growth of our man-made distributed systems, we wonder how the ants, bees, birds, fish, and higher animals have managed to perform so well with their distributed systems. If we are ever to achieve a level of performance anywhere near theirs, we will have to further uncover the underlying principles of distributed-systems behavior (see sidebar). We have discussed some of these in this article, but there is much new ground to be broken. Almost anywhere you dig you are likely to find pay dirt. The field is wide open for new ideas and new approaches, challenging problems remain unsolved, and the application of new results will be widespread and rapid—what lovelier environment could you seek?

REFERENCES

1. Amdahl, G.M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of AFIPS*, Vol. 30. Thompson, Washington, D.C., 1967, pp. 483–485.
2. Belghith, A., and Kleinrock, L. Analysis of the number of occupied processors in a multi-processing system. UCLA CSD Rep. 850027, Computer Science Dept., Univ. of California, Los Angeles, Aug. 1985.
3. Denning, P.J. Parallel computation. *Am. Sci.* (July–Aug. 1985).
4. Ein-Dor, P. Grosch's law re-revisited: CPU power and the cost of computation. *Commun. ACM 28*, 2 (Feb. 1985), 142–151.
5. Ercegovac, M., and Lang, T. General approaches for achieving high speed computations. In *Supercomputers*, S. Fernbach, Ed. North-Holland, Amsterdam. To be published.
6. Gerla, M., and Kleinrock, L. Flow control protocols. In *Computer Network Architectures*, Paul Green, Ed. Plenum, New York, 1982, pp. 361–412.
7. Grosch, H.A. High speed arithmetic: The digital computer as a research tool. *J. Opt. Soc. Am. 43*, 4 (Apr. 1953).
8. Grossberg, S. *Studies of the Mind and Brain: Neural Principles of Learning, Perception, Development Cognition and Motor Control*. Reidel, Hingham, Mass., 1982.
9. Hwang, K. Multiprocessor supercomputers for scientific/engineering applications. (June 1985), 57–73.
10. Hwang, K., and Briggs, F. *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, 1984.
11. Kleinrock, L. *Queueing Systems, Volume 2: Computer Applications*, Chap. 4. Wiley-Interscience, New York, 1976.
12. Kleinrock, L. On flow control in computer networks. In *IEEE Proceedings of the Conference in Communication*, Vol. 2. IEEE, New York, June 1978, pp. 27.2.1–27.2.5.
13. Kleinrock, L. On the theory of distributed processing. In *Proceedings of the 22nd Annual Allerton Conference on Communication, Control and Computing*, Univ. of Illinois, Monticello, Oct. 1984, pp. 60–70.
14. Kuck, D.J. et al. The effects of program restructuring, algorithm change and architecture choice on program. In *Proceedings of the International Conference on Parallel Processing*, Aug. 1984, pp. 129–138.
15. Minsky, M., and Papert, S. On some associative, parallel and analog computations. In *Associative Information Technologies*, E.J. Jacks, Ed. Elsevier North Holland, New York, 1971.
16. Patton, C.P. Microprocessors: Architecture and applications. *IEEE Comput. Mag. 18*, 6 (June 1985), 29–40.
17. Schneck et al. Parallel processor programs in the federal government. *IEEE Comput. Mag. 18*, 6 (June 1985), 43–56.
18. Shannon, C., and Weaver, W. *The Mathematical Theory of Communication*. Univ. of Illinois Press, Urbana, 1962.
19. Stuck, B.W., and Arthurs, E. *A Computer and Communications Network Performance Analysis Primer*. Prentice-Hall, Englewood Cliffs, N.J., 1985.
20. Tsetlin, M.L. *Automaton Theory and Modeling of Biological Systems*. Academic Press, New York, 1973.
21. Yemini, Y. A statistical mechanics of distributed resource sharing mechanisms. In *Proceedings of INFOCOM 83*, 1983, pp. 531–539.

CR Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Design, Performance, Theory

Additional Key Words and Phrases: computer networks, distributed control, distributed processing, economics of computing power, machine architecture, multiprocessing, parallel processing, performance of distributed systems, self-organizing systems

Author's Present Address: Leonard Kleinrock, Dept. of Computer Science, Boelter Hall, UCLA, Los Angeles, CA 90024.

Teletraffic Analysis and Computer Performance Evaluation
O J Boxma, J W Cohen, H.C. Tijms (Editors)

PERFORMANCE MODELS FOR DISTRIBUTED SYSTEMS

Leonard Kleinrock

Computer Science Department
University of California, Los Angeles
Los Angeles, California
U.S.A.

We discuss the relatively sad state of affairs regarding performance evaluation of distributed systems. We introduce some of the popular models, discuss the architecture and algorithm modelling, and then proceed to present a scattering of known results. Finally, a number of open areas are discussed which offer directions through which we hope to gain more understanding and insight into the operation and principles of these systems.

1. INTRODUCTION

We have finally seen the convergence of two giant technologies, namely, data processing and data communications. For years they had each been developing individually at a breakneck pace fueled largely by the needs demanded by the end user and by the enormous cost/performance improvements brought about by the semiconductor industry. Now they have merged, and they have created a technological environment whose potential is staggering.

The environment is one which is heavily distributed. We are beginning to see systems with distributed communications, distributed storage, distributed processing and distributed control. We are moving headlong into such structures in response to the user demands and in response to the manufacturers' desire to roll out products as rapidly as possible.

Unfortunately, we lack an understanding of the basic principles of distributed systems, principles which are badly needed to predict performance, to explain behavior and to establish design methodologies. We are even lacking the basic models to describe these systems. And the systems we are constructing are running into various performance problems in the form of high cost, poor response time, low efficiency, deadlocks, difficulty in growth, unproved operational algorithms, etc.

We are implementing too quickly for the theory, on the one hand, and too slowly for the applications, on the other. The situation is frustrating, due, in part, to the rapid rate at which technological change is occurring relative to the ability of our analytical science to keep pace with it. Even more frustrating is the recognition that Nature seems to have implemented some exquisite distributed systems which defy our understanding and analysis; further, these systems appear to be structured very very differently from the way we build our distributed systems today.

In this paper, we discuss some of the issues and performance measures, comment on architectures and algorithms, present some of the meager and scattered results which have been established, and then offer some open areas and problems.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-82-C0064.

2. ISSUES AND PERFORMANCE MEASURES

A distributed system consists of sources which are distributed and which create demands for service from a set of resources which themselves are distributed. As usual, more than one source may demand service from the same resource at the same time, causing a conflict. As queueing theorists, we know how to resolve such conflicts: we ask the demands to form an orderly queue to be served in some order, usually one at a time. This turns out to be a wonderfully efficient mechanism for resolving the conflict; indeed, it implements the demand access procedure known in the field of telecommunications as "statistical multiplexing."

Unfortunately, in a distributed environment, one cannot form a queue without incurring some overhead costs; this is the case because an individual source cannot automatically "see" who is on the queue and must therefore devote some effort to discover this information. This gives rise to the study of multiaccess algorithms, a field which has received some attention in the last few years. In spite of this work, there is still no comprehensive theory of multiaccess algorithms which explains the fundamental principles and observed phenomena. We will not devote much effort to that subject in this paper. Instead, we will focus here on the area of distributed processing. (The long term goal is still to understand the interaction of communication, storage, processing and control, but that global understanding is not yet within our grasp.)

One of the major issues in distributed processing is that of concurrency. Concurrency is a measure of the number of resources which can be utilized simultaneously. For example, in computer networks, a number of different communication channels can be transmitting messages at the same time. In packet radio systems, one introduces the notion of spatial reuse of a channel.

In distributed processing, the concurrency we discuss refers to the number of processors which can be active at a given time. The average number of busy processors is known as the "speedup," S, for a given problem. S measures how much faster a job can be processed using multiple processors, as opposed to using a single processor.

Another key measure of interest is the efficiency with which the processing resources are being used (i.e., the utilization factor). The utilization factor is proportional to the throughput, γ, of the system. The throughput (or carried traffic) is a function of the blocking probability, B, and the offered traffic, λ, through the relationship $\gamma = \lambda B$.

Clearly, the mean response time, T, is a major performance measure for our systems. The mean-response-time-versus-throughput profile is perhaps one of the most common and important profiles used in the study of congestion systems. Therefore, we see that the speedup, S, is also the ratio of the mean response time using a single processor to the mean response time using multiple processors.

It has been found convenient to define a performance measure which combines all of these other measures into a single measure which we call "power." The power, P, is defined as

$$P = \frac{\gamma}{T}$$

Below, we discuss some of the fascinating properties enjoyed by this particular performance measure.

In quoting the results regarding distributed processing, we must distinguish a number of cases. We must specify whether we are talking about the response time and concurrency for a single job, or for a fixed number of jobs all of which are waiting for processing, or for a stream of jobs which arrive at random times. We must identify the number of processors which are available to work on the collection of jobs. We must know the way in which processors are assigned to jobs (i.e., scheduling and assignment). We must know if communication is instantaneous or congested, if it is unicast, multicast or broadcast, and we must know the communications connectivity among the

various system resources (e.g., processors and storage). The system description can easily get far more complex than our meager analytic tools can handle at present.

3. ARCHITECTURES AND ALGORITHMS

The world of applications, especially scientific applications, has an insatiable appetite for computing power. Any good mathematician can consume any finite computing capability by posing a combinatoric problem whose computational complexity grows exponentially with a variable of the problem (e.g., the enumeration of all graphs with N nodes). The ways in which we push back this "power wall" involve both hardware and software solutions. Typically, the methods for speeding up the computation include the following:

- faster devices (a physics and engineering problem)
- architectures which permit concurrent processing (a system design problem)
- optimizing compilers for detecting concurrency (a software engineering problem)
- algorithms for specification of concurrency (a language problem)
- more expressive models of computation (an analytic problem).

3.1 ARCHITECTURES

There are many ways of classifying machine architectures --- too many, in fact. We select the following classification, which we feel is appropriate for the purposes of this paper.

We begin with the purely serial uniprocessor in which a single instruction stream operates on a single data stream (SISD). These systems are "centralized" at the global level but really do contain many elements of a distributed system at the lower levels, for example, at the level of communications on the VLSI chips themselves.

Next is the vector machine, in which a single instruction stream operates on a multiple data stream (SIMD). These include array processors (e.g., systolic arrays) and pipeline processors.

The third member consists of multiple processors which, collectively, can process multiple instruction streams on multiple data streams (MIMD). When the multiple processors cooperate closely to process tasks from the same job, we usually refer to this form of multiprocessing as parallel processing. On the other hand, the term distributed processing is applied to the form of multiprocessing that takes place when the multiple processors cooperate loosely and process separate jobs.

Vector machines and the multiprocessing systems all provide some form of concurrency. The effect of this concurrency on system performance is important and is therefore a very active area of research.

Since the onslaught of the VLSI revolution, a number of machine architectures have been implemented in an attempt to provide the supercomputing power toward which concurrent processing tempts us [1]. Two excellent, recent summaries of some of these projects are offered by Hwang [2] and by Schneck et al. [3].

3.2 ALGORITHMS AND MODELS OF COMPUTATION

The major goal in characterizing the algorithm is to identify and exploit its inherent parallelism (i.e., potential for concurrency). There are many levels of resolution at which we can attempt to find this parallelism, and we list these levels below in decreasing order of granularity [4].

- job execution
- task execution
- process execution
- instruction execution
- register-transfer
- logic device

Clearly, as we drop down the list to finer granularity, we expose more and more parallelism, but we also increase the complexity of scheduling these tiny objects to the processors and of providing the communications among so many objects (the problem of interprocess communication --- IPC). As was stated earlier, if we operate at the top level (i.e., at the job level), then we think of the system as a distributed processing system; if we operate at the task or process level, then we have a parallel processing system; if we operate at the instruction level, we have the vector machine and the array processor.

Regardless of the level at which we operate, it behooves us to create a "model" of the algorithm or, if you will, the computation we are processing. A very common model is the graph model of computation (the task graph) [5], which is normally used at the task or process level. In this model, the nodes represent the tasks (or processes), and the directed edges represent the dependencies among the tasks, thereby displaying the partial ordering of the tasks and the parallelism which can be exploited. See Figure 1 for an example.

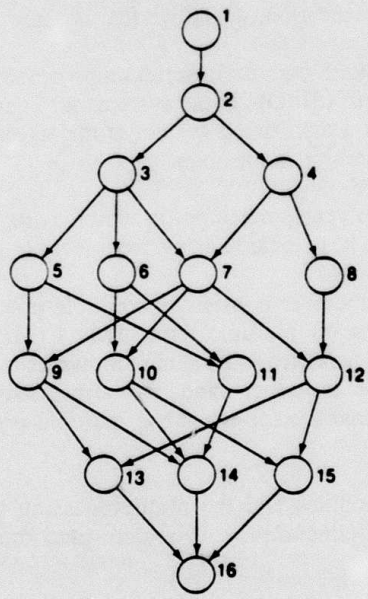


Figure 1
The Graph Model of Computation

Another common model of algorithms is the Petri Net [5]. See Figure 2 for an example. In this model, a bipartite directed multigraph is created with two sets of nodes, each of a different type.

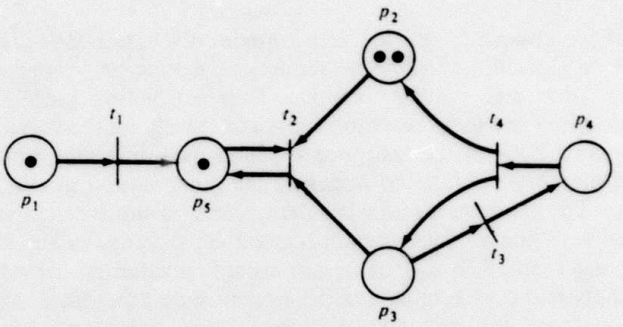


Figure 2
The Petri Net Model

Figure taken from Peterson, J.L., Petri Net Theory and the Modeling of Systems (Prentice-Hall, Englewood Cliffs, N.J., 1981, p. 17, Figure 2.11).

The first kind of node is called a place, say p_i, and is drawn as a circle; it represents a condition. The second kind of node is called a transition, say t_i, and is drawn as a bar perpendicular to its incident arcs; it represents a change in state (an event). The directed arcs represent dependencies between the events and the conditions on the nodes they connect. Tokens are located in certain of the places and such a token distribution is called a marking; the marking identifies the state of the system. The graph is initialized by placing a number of tokens at certain places in the graph. When all branches leading into a transition node contain at least one token (this transition has all of its conditions satisfied), then the transition "fires"; this firing action removes one token from each of the places feeding this transition (if more than one transition can fire, one is chosen at random) and puts one token in each place fed by the transition. In Figure 2, we can see that t_1 is ready to fire. The process continues to operate as satisfied transitions fire and tokens travel around the graph, causing the system to move from one state to another. Such models are often used to check for proper termination of algorithms. A number of extensions of Petri Nets have been developed. One major extension has been to introduce the dimension of time by labelling each transition with a quantity representing the time it takes for the transition to occur. Timed Petri Nets [6], (TPN), assume deterministic values for the transition delays. Stochastic Petri Nets [7], (SPN), allow for random transition times and lead to a Markov Chain analysis. Petri Nets are capable of modelling parallelism, conflict and synchronization of processes. An excellent document summarizing much of the recent work on Timed Petri Nets may be found in the Proceedings of the International Workshop on Timed Petri Nets (July, 1985) [8]. Unfortunately, one quickly gets entangled in the state space explosion problem with these models.

We will focus mainly on the former model (the graph model of computation) in this paper. Note that one may allow the graph model for a job to be drawn randomly from a set of possible graphs. Furthermore, we must specify the distribution of time required to process each task in the graph.

3.3 MATCHING THE ARCHITECTURE TO THE ALGORITHM

The performance of a distributed system depends strongly on how well the architecture and the algorithm are matched. For example, a highly parallel algorithm will perform well on a highly

parallel architecture; a distributed system requiring lots of interprocessor communication will perform poorly if the communication bandwidth is too narrow. This matching problem becomes fierce and crucial when we attempt to coordinate an exponentially growing number of processors requiring an exponentially growing amount of interprocessor communication. The apparent solution to such an unmanageable problem is one that is self-organizing.

If we choose to use the graph model discussed above, then we are faced with a number of architecture/algorithm problems, namely, partitioning, scheduling, memory access, interprocess communication and synchronization. The partitioning problem refers to the decisions we must make regarding the level of granularity and the choices involving which objects should be grouped into the same node of the task graph. The scheduling problem refers to the assignment of processors and memory modules to nodes of the computation graph. In general, this is an NP-complete problem. The memory access problem refers to the mechanism provided for processors to communicate with the various memory modules; usually, either shared memory or message-passing schemes are used. The interprocessor communication problem refers to the nature of the communication paths and connections available to provide processors access to the memory modules and to other processors; this may take the form of either an interconnection network in a parallel processing system, a local area network in a local distributed processing or shared data or shared peripheral system, or a packet-switched value-added long-haul network in a nationwide distributed system. Synchronization refers to the requirement that no node in the graph model can begin execution until all of its predecessor nodes have completed their execution.

The use of broadcast or multicast communication opens up a number of interesting alternatives for communication. Local area networks take exquisite advantage of these communication modes. Algorithms which require tight coupling (i.e., lots of IPC) need not only large bandwidths (which, for example, could be provided by fiber-optic channels), but also low latency. Specifically, the speed of light introduces a 15,000 microsecond latency delay for a communication which must travel from coast to coast across the United States.

Another consideration in matching architectures to algorithms is the balance and trade-off among communication, processing and storage. We have all seen systems where one of these resources can be exchanged for the others. For example, if we do some preprocessing in the form of data compression prior to transmission, we can cut down on the communication load (trade processing for communication). If we store a list of computational results, we can cut down on the need to recompute the elements of the list each time we need the same entry (trade storage for processing). Similarly, if we store data from a previous communication, we need merely transmit the data address or name of the previous message rather than the message itself (trade storage for communication). Selecting the appropriate mix in a given problem setting is an important issue.

Distributed algorithms operating in a distributed network environment (e.g., a packet switched network) face the problem that network failures may cause the network to temporarily be partitioned into two (or more) isolated subnetworks. In such a case, detection and recovery mechanisms must be introduced.

Lastly, it should be mentioned that very little is known about characterizing those properties of an algorithm which cause them to perform well or poorly in a distributed environment.

4. RESULTS

We do know some things about the way distributed systems behave, precious few though they may be. The most interesting thing about them is that they come to us from research in very different fields of study. Unfortunately, the collection of results (of which the following is a sample) is just that --- a collection, with no fundamental models or theory behind it.

We begin by considering closely coupled systems, in particular, parallel processing systems. One of the most compelling applications of parallel processing is in the area of scientific computing

where the speed of the world's largest uniprocessors is hopelessly inadequate to handle the computational complexity required for many of these problems [9]. Of course, the idea is that, as we apply more parallel processors to the computational job, then the time to complete that job will drop in proportion to the number of processors, P.

4.1 SPEEDUP

Recall the definition of the speedup factor, S. The best we can achieve is for S to grow directly with P, that is,

$$S \leq P$$

Thus, in general, $1 \leq S \leq P$. In the early days of parallel processing, Minsky [10] conjectured a depressingly pessimistic form for the typical speedup, namely,

$$S = \log P$$

Often, that kind of poor performance is, indeed, observed. Fortunately, however, experience has shown that things need not be that bad. For example, we can achieve $S = 0.3\,P$ for certain programs by carefully extracting the parallelism in Fortran DO loops [11]. However, Amdahl has pointed out a serious limitation to the practical improvements one can achieve with parallel processing (the same argument applies to the improvements available with vector machines) [12]. He argues that if a fraction, f, of a computation must be done serially, then the fastest that S can grow with P is

$$S \leq \frac{P}{[Pf + 1 - f]}$$

We see that, for $f = 1$ (everything must be serial), $S_{\max} = 1$; for $f = 0$ (everything in parallel), then $S_{\max} = P$.

The actual amount of parallelism (i.e., S) achieved in a parallel processing system is a quantity which we would like to be able to compute. S is a strong function of the structure of the computational graph of the jobs being processed. I, with one of my students [13], have been able to calculate S exactly, as a simple function of the graph model. Specifically, we consider a parallel processing system with P processors and with an arrival rate of λ jobs per second. We assume the collection of jobs can be modelled with an arbitrary computation graph with an average of N tasks per job, each task requiring an average of $\bar{x}$ seconds. Then it can be shown that

$$S = \begin{cases} \lambda N \bar{x} & \text{for} \quad \lambda N \bar{x} \leq P \\ P & \text{for} \quad \lambda N \bar{x} \geq P \end{cases}$$

This is a very general result, and is really based on the definition of the utilization factor; in some special cases, the distribution of the number of busy processors can be found, as well.

4.2 SOME CONFIGURATION ISSUES

So far, we have given ourselves the luxury of increasing the system's computational capacity as we have added more processors to the system. Let us now consider adding more processors, but in a fashion which maintains a constant total system capacity (i.e., a constant system throughput in jobs completed per second). This will allow us to see the effect of *distributing* the computation for a job over many smaller processors. The particular structure we consider is the regular series-parallel structure shown in Figure 3, where we have taken a total processing capacity of C MIPS (million instructions per second) and divided it equally into mn processors, each behaving as an M/M/1 queue, and each of C/mn MIPS. On entering the system, a job selects (equally likely) any one of the m series branches down which it will travel. It will receive $1/n$ of its total processing needs at each of the n series-connected processors.

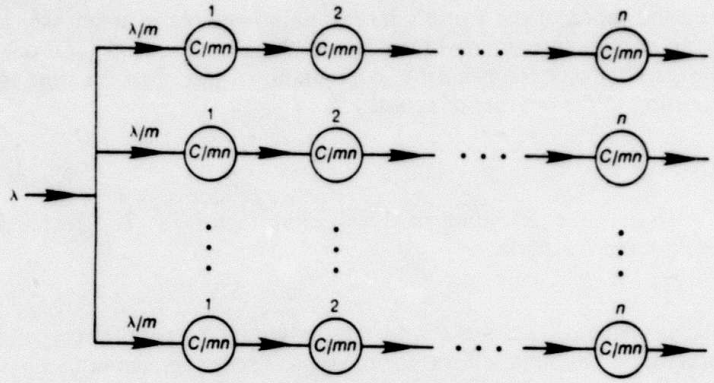


Figure 3
A Symmetrical Distributed-Processing Network

The key result for this system [14] is that the mean response time for jobs in this series-parallel pipeline system is mn times as large as it would have been had the jobs been processed by a single processor of C MIPS! The message is clear --- distributed processing of this kind is terrible. Why, then, is everyone talking about the advantages of distributed processing? The answer must be that a large number of small processors (e.g., microprocessors) with an aggregrate capacity of C MIPS is less expensive than a large uniprocessor of the same total capacity. It can be shown that the series-parallel system *will* have the same response time as the uniprocessor if the aggregrate capacity of the series-parallel system has K times the capacity of the uniprocessor, where

$$K = mn - \rho(mn - 1)$$

and where ρ is the utilization factor for each processor; namely, ρ = arrival rate of jobs times the average service time per job for a processor. This says that, for light loads $(\rho \ll 1)$, $K = mn$, whereas, for heavy loads $(\rho \to 1)$, $K = 1$. Is it the case that smaller machines are mn times as cheap as large machines (so that we can purchase mn times the capacity at the same total price, as is needed in the light-load case)? To answer this question, recall a law that was empirically observed by Grosch more than three decades ago. Grosch's Law [15] states that the capacity of a computer is related to its cost, which we denote by D (dollars), through the following equation:

$$C = JD^2$$

where J is a constant. This law may be rewritten as

$$\frac{D}{C} = \frac{1}{\sqrt{JC}}$$

Grosch tells us that the economics are *exactly the reverse* of what we need to break even with distributed processing! He says that larger machines are cheaper per MIPS. If Grosch is correct today, then why are microprocessors selling like hotcakes? A more recent look at the economics explains why. Ein-Dor [16] shows that, if we consider all computers at the same time, Grosch's Law is clearly not true, as seen in Figure 4. In this figure, we see that microcomputers are a good buy. However, as Ein-Dor points out, Grosch's Law is still true today if we consider *families* of computers. Each family has a decreasing cost per unit of capacity as capacity is increased. Ein-Dor goes on to make the observation that, if one needs a certain number of MIPS, then one should purchase computers from the smallest family that currently can supply that many MIPS. Furthermore, once in the family, it pays to purchase the biggest member machine in that family (as predicted by Grosch).

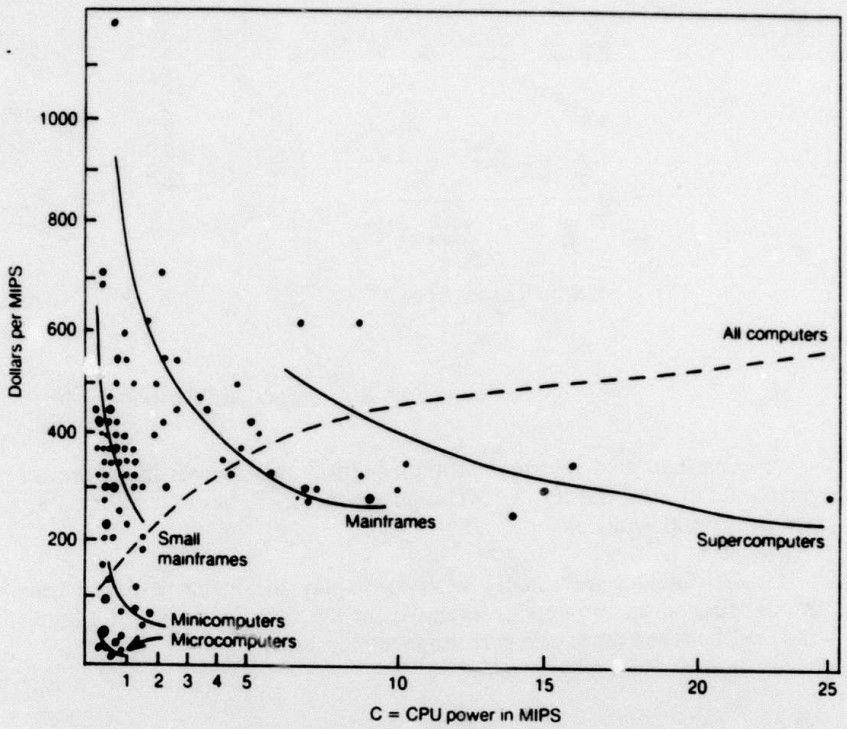
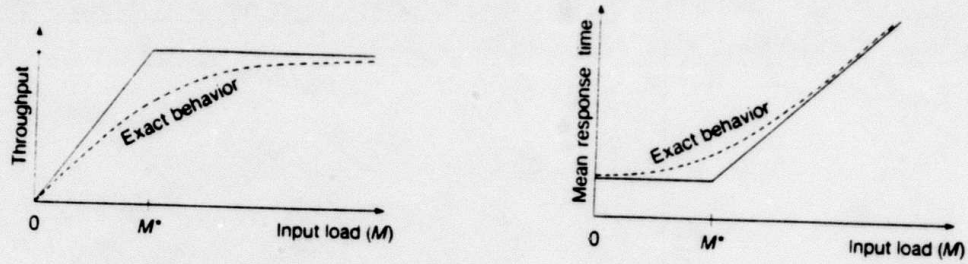


Figure 4
Economics of Computer Power

Figure taken from Ein-Dor, P., Grosch's law re-visited: CPU power and the cost of computation, Commun. ACM 28, 2 (Feb., 1985), 142-151.

4.3 DEADLOCKS AND POWER

Now that we have discussed the performance of parallel processing systems for some special cases, let us generalize the ways in which jobs pass through a multi-processor system, and analyze the system throughput and response time. Indeed, we bound these key system performance measures in the following way: Suppose we have a population of M customers competing for the resources of the system. Assume that customers generate jobs which must be processed by certain of the system resources, that the way in which these jobs bounce around among the resources is specified in a probabilistic fashion and that the mean response time of this system is T seconds. When a customer's job leaves the system, that customer then begins to generate another job request for the system, where the average time to generate this request is t_0 seconds. Of interest is the mean response time, T, and the system throughput γ as a function of the other system parameters. Although we have been extremely general in the system description, we can nevertheless place an excellent upper bound on the system throughput and an excellent lower bound on the mean response time, as shown in Figure 5. In this figure, the quantity M^* is defined as the ratio of the mean cycle time $T_0 + t_0$ to the mean time x_0 required on the critical resource in a cycle; T_0 is the mean response time when $M = 1$, and the critical resource is that system resource that is most heavily loaded [17]. To find the exact behavior (as shown in dashed lines in the figure) rather than the bounds, one must specify the distributions of the service time required by jobs at each resource in the system as well as the queueing discipline at each. That is, we must set up the closed queueing network model. Using the bounds or the exact results, one can easily see the effect of parameter changes on the system behavior. For example, one can examine the accuracy of the common rule-of-thumb which suggests that the proper mix of microprocessor speed, memory size and communication bandwidth is in the proportion 1 MIPS, 1 megabyte and 1 megabit-per-second; some suggest that we will soon see a 10,10,10 mix instead of this 1,1,1 mix.



(a) Bound on Throughput

(b) Bound on Mean Response Time

Figure 5
Bounds on Throughput and Response Time

One can also interpret profiles such as that shown in Figure 5a as the carried traffic (γ) versus the offered traffic (λ). This function, $\gamma(\lambda)$, is often referred to as the "flow control" function in systems analysis.

Of course, we usually wish to display the mean response time, T, as a function of the throughput, γ, as shown, for example, in the familiar curve of Figure 6. As usual, we note the trade-off between these two performance measures.

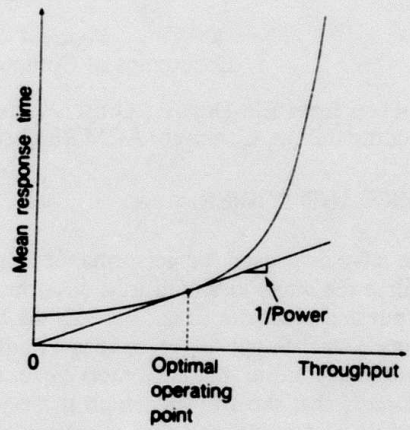


Figure 6
The Key System Profile and Power

We are immediately compelled to inquire about the location of the "optimal" operating point for a system. The answer depends upon how much you hate delay versus how much you love throughput. One way to quantify this love-hate choice is to adopt our earlier definition of power, $P = \frac{\gamma}{T}$, as our objective function. The operating point that optimizes (i.e., maximizes) the power (large throughput and small delay) is located at that throughput such that

$$\frac{dT}{d\gamma} = \frac{T}{\gamma}$$

when the derivative exists. Even in the discontinuous case, this optimum occurs where a straight line (of minimum slope) out of the origin touches the throughput-delay profile (usually tangen-

tially); such a tangent and operating point are shown in Figure 6. *This result holds for all profiles and all flow control functions*. Moreover, for all M/G/1 systems, this optimal operating point implies that the system should be loaded in such a way that each resource has, on the *average, exactly one job* to work on [18]. In the case of an M/M/1 queue, the optimum power point occurs at half the maximum throughput (50% efficient) and twice the minimum delay. It turns out that the average number in system is a natural invariant to consider when one maximizes power.

5. OPEN AREAS

In the previous section, we listed a few of the known results in performance evaluation of distributed systems. Other results do exist, and they come from many diverse fields. Nevertheless, one has the uneasy feeling, as stated earlier, that we have only scratched the surface. The underlying results have still not been uncovered. Much work needs to be done.

In this section, we present some intriguing results and ideas which perhaps will lead to some fundamental understanding.

A large general area of interest is that of distributed control systems. One example of distributed control is the dynamic routing procedure found in many of today's packet switching networks where no single switching node is responsible for the network routing. Instead, all nodes participate in the selection of network routes in a distributed fashion. A great deal of research is currently under way to evaluate the performance of other distributed algorithms in networks and distributed systems. Examples are the distributed election of a leader, distributed rules for traversing all the links of a network and distributed rules for controlling access to a database.

Another large class of distributed control algorithms has to do with sharing a common communication channel among a number of devices in a distributed fashion [19]. If the channel is a broadcast channel (also known as a one-hop channel), then the analytic and design problem is fairly manageable and a number of popular local area network algorithms for media access control have been studied and implemented. Examples here include CSMA/CD (carrier-sense multiple access with collision detect --- as used in Xerox's Ethernet, AT&T's 3B-Net and Starlan and IBM's PC Network), token passing (as used in the IBM Token-Ring and Token-Bus networks) and address contention resolution (as used in AT&T's ISN). A large number of additional channel access algorithms have been studied in the literature including, for example, Expressnet, tree algorithms, urn models, and hybrid models. If the channel is multicast (or multi-hop), then the analytic problem becomes much harder.

But what if the processors in our distributed environment are allowed to communicate with their peers in very limited ways? Can we endow these processors (let us call them automata for this discussion) with an internal algorithm which will allow them to achieve a collective goal? Tsetlin [20] studied this problem at length and was able to demonstrate some remarkable behavior. For example, he describes the Goore game in which the automata possess finite memory and act in a probabilistic fashion based on their current state and the current input; they cannot communicate with each other at all and are required to vote YES or NO at certain instants of time. The automata are not aware of each other's vote; however, there is a referee who can observe and calculate the percentage, p, of automata that vote YES. The referee has a function, $f(p)$ (such as that shown in Figure 7), where we require that $0 \le f(p) \le 1$. Whenever the referee observes a percentage, p, who vote YES, he will, with probability $f(p)$, reward each automaton, independently, with a one dollar payment; with probability $1 - f(p)$ he will punish an automaton by taking one dollar away. Tsetlin proved that no matter how many players there may be in a Goore game, if the automata have sufficient memory, then for the payoff probability shown in the figure, exactly 20% of the automata will vote YES with probability one! This is a beautiful demonstration of the ability of a distributed processing system to act in an optimum fashion, even when the rules of the reward function are unknown to the players and when they can neither observe nor communicate with each other. All they are allowed is to vote when asked, and to observe the reward or penalty they receive as a result of that vote. In this work we see the beginnings of a theory which may be able to explain collective phenomena.

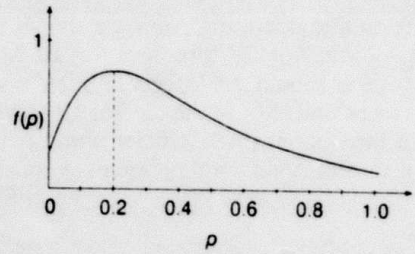


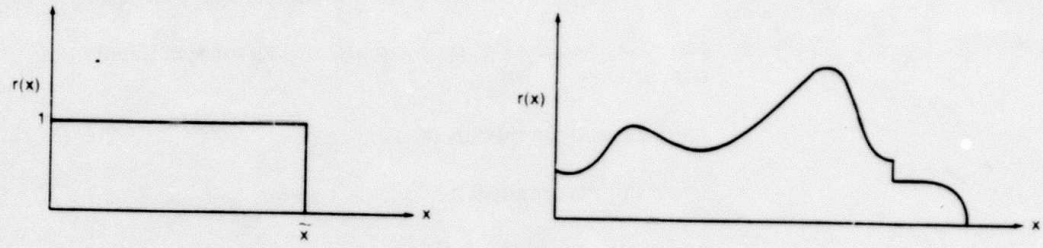
Figure 7
The Goore Game

Another fascinating study of late has to do with the subject of "common knowledge" [21]. It is best exemplified by the problem of two friendly armies camped on opposite hills overlooking an enemy army camped in the valley between them. Neither of the friendly armies can defeat the enemy by itself, but both together surely can. The general of the first friendly army ("army A") sends a messenger to the general of the second friendly army ("army B") one night, informing him that they are to attack the enemy at dawn. Unfortunately, the messenger must travel through the enemy valley and may not reach his destination. Now, suppose the messenger is successful. Will army B's general attack at dawn? Clearly not, since he must acknowledge to army A that he got the message in order for army A to be willing to attack. So he sends the messenger back; let us assume the messenger makes it again. Now will army A attack? No, since army B is not sure that they (army A) got the acknowledgement and if they did not get it, army B should not proceed. And so it continues, with acknowledgements of acknowledgements of acknowledgements, etc, ad infinitum. Neither army will ever attack! Neither has common knowledge which is defined to be knowledge that A knows that B knows that A knows that B knows .... all the way. Common knowledge is an important concept in distributed systems. In fact, it is related to our earlier comment in Section 2: the problem of forming a queue in a distributed environment is one of common knowledge, to a degree.

Most of us know how to play the game of 20 questions. Suppose I asked you to determine which number I was thinking of from the set 0,1,2,...,15. You could clearly determine the answer with only four yes/no questions by first determining which half of those contained my selection by asking if the number was less than 8. Based on the answer to the first question, you would then split the eligible 8 into two sets of 4, etc, finally converging on the answer in 4 questions. Now, could you simply ask four questions in parallel (i.e., write down the four questions to be answered)? Specifically, you could not make any of the questions depend on the answer to any of the others. The answer is yes (and it is trivial to see - it is left as a fun exercise for the reader). The fact that it is possible says that this type of problem lends itself very nicely to parallel processing with lots of concurrency. Some recent work has been reported in this area in [22].

The degree of coupling between processes affects their ability to take advantage of multiple processors. If they are completely uncoupled, then one can utilize as many processors as one has processes. If each process depends upon the other in some sequential fashion, the amount of concurrency may be severely limited. The models described in Section 3.2 expose some of this coupling. The challenge is to find a proper, useful measure of coupling among processes that can be used to predict the speedup they can achieve.

We have mentioned the trade-off between processing and communications. Indeed, we have been able to show [23] that the use of broadcast communications can assist significantly in the following situation. Suppose we have K sorted sublists in each of K distributed processors and we wish to merge and sort the entire list. There is a simple way to transmit the sorted sublists using broadcast communications in such a way that the receiving processors will accept the list elements in their properly sorted order; thus there will be no global sorting required after the merging (i.e., the transmission) has taken place.



(a) Classical Queueing (b) New Queueing Class

Figure 8
Desired Rate of Service as a Function of the Elapsed Time in Service

We close this section by introducing a new class of queueing problem which the author and two of his students (Abdelfettah Belghith and Jau-Hsiung Huang) have been studying with relatively little analytical success. Consider the task graph shown in Fig 1. Let us assume that each task requires a random amount of time to be processed (for simplicity, let us assume that each is exponentially distributed with rate μ. Further, assume that no more than one processor may work on a given task. When node (task) 1 is being processed, the job proceeds at a rate μ. When it completes, node 2 begins and again the job proceeds at a rate μ. When it finishes, nodes 3 and 4 both become activated and, assuming at least two processors are available, the job then proceeds at a rate 2μ, etc. What we see is that the rate at which a job can absorb work depends upon where it is in its processing cycle (i.e., which tasks are currently being processed) and also depends upon how many processors are available to work on the job (other jobs may be competing for a finite number of processors). Of course, we would like to handle the case of an arbitrary distribution of task times. The classic queueing model, shown in Figure 8a, assumes that the rate at which a job can absorb work is constant, and that the total amount of work is a random variable, $\tilde{x}$. Specifically, we plot $r(x)$, the desired rate of service (seconds per second) as a function of the elapsed time in service (assuming no competition for service). The total service time is the area under the curve. In Figure 8b, we show a general picutre for our new class of problem where the rate varies with elapsed time.

6. CONCLUSIONS

An overall theory and understanding is lacking for distributed systems behavior. We need considerably sharper tools to evaluate the ways in which randomness, noise and inaccurate measurements affect the performance of distributed systems. What is the effect of distributed control in an environment where that control is delayed, based on estimates and not necessarily consistent throughout the system? What is the effect on performance of scaling some of the system parameters? We need a common metric for discussing the various system resources of communications, storage and processing. For example, is there a processing component to communications? We need a proper way to discuss distributed algorithms and distributed architectures.

A microscopic theory that deals with the interaction of each job with each component of the system is likely to overwhelm us with detail and fail to lead us to an understanding of the overall system behavior. It is similar to the futility of studying the many-body problem in physics in order to obtain the global behavior of solids. What is needed is a macroscopic theory of distributed systems, much as thermodynamics has provided for the physicist. In fact, Yemini [24] has proposed an approach for a macroscopic theory based upon statistical mechanics that will lead to better understanding the global behavior of distributed systems without a detailed, fine-grained analysis.

Massively distributed and massively connected systems with enormous computational capacity are likely to appear in the next ten years. Some of the directions which are needed to assist in the analysis and design of these systems are the following:

- developing innovative architectures for parallel processing
- providing better languages and algorithms for specification of concurrency
- more expressive models of computation
- matching the architecture to the algorithm
- understanding the tradeoff among communication, processing and storage
- evaluation of the speedup factor for classes of algorithms and architectures
- evaluation of the cost-effectiveness of distributed processing networks
- study of distributed algorithms in networks
- investigation of how loosely coupled self-organizing automata can demonstrate expedient behavior
- development of a macroscopic theory of distributed systems
- understanding how to average over algorithms, architectures and topologies to provide meaningful measures of system performance

It is through developments such as these that we hope to establish a unified theory for distributed systems.

The author takes pleasure in acknowledging the expert assistance and unlimited energy of Jodi L. Feiner in preparing this manuscript for publication. The many hours she devoted to this arduous task have helped produce this document.

REFERENCES

[1] Ercegovac, M., and Lang, T., General approaches for achieving high speed computations, Supercomputers, North-Holland, Amsterdam, to be published.

[2] Hwang, K., Multiprocessor supercomputers for scientific/engineering applications (June, 1985), 57-73.

[3] Schneck et al., Parallel processor programs in the federal government, IEEE Computer Magazine 18, 6 (June 1985), 43-56.

[4] Patton, C.P., Microprocessors: architecture and applications, IEEE Computer Magazine 18, 6 (June 1985), 29-40.

[5] Peterson, J.L., Petri Net Theory and the Modeling of Systems (Prentice-Hall, Englewood Cliffs, N.J., 1981).

[6] Ramchandani, C., Analysis of asynchronous concurrent systems by time Petri nets, Project MAC Technical Report MAC-RT-120, MIT, (1974).

[7] Molloy, M.K., Performance analysis using stochastic Petri nets, IEEE Trans. on Computers, vol. 31, no. 9 (1982) 913-917.

[8] Proceedings of the International Workshop on Timed Petri Nets (IEEE Computer Society Press, Maryland, 1985).

[9] Denning, P.J., Parallel computation, Am. Sci., (July 1985).

[10] Minsky, M. and Papert, S., On some associative, parallel and analog computations, in: Jacks, E.J. (ed.), Associative Information Technologies, (Elsevier North-Holland, New York, 1971).

[11] Kuck, D.J., et al., The effects of program restructuring, algorithm change and architecture choice on program, in: Proceedings of the International Conference on Parallel Processing (1984).

[12] Amdahl, G.M., Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of AFIPS, Vol. 30, (1967).

[13] Belghith, A. and Kleinrock, L., Analysis of the number of occupied processors in a multiprocessing system, UCLA CSD Rep. 850027, Computer Science Dept., Univ. of California, Los Angeles, (August 1984).

[14] Kleinrock, L., On the theory of distributed processing, in: Proceedings of the 22nd Annual Allerton Conference on Communication, Control and Computing, Univ. of Illinois, Monticello, October 1984.

[15] Grosch, H.A., High speed arithmetic: The digital computer as a research tool, J. Opt. Soc. Am. 43, 4 (April 1953).

[16] Ein-Dor, P., Grosch's law re-revisited: CPU power and the cost of computation, Commun. ACM 28, 2 (Feb. 1985).

[17] Kleinrock, L., Queueing Systems, Volume 2: Computer Applications, Chap. 4. (Wiley Interscience, New York, 1976).

[18] Kleinrock, L., On flow control in computer networks, in: IEEE Proceedings of the Conference in Communication, Vol. 2, IEEE, New York, June 1978.

[19] Stuck, B.W. and Arthurs, E., A Computer and Communications Network Performance Analysis Primer (Prentice-Hall, Englewood Cliffs, N.J., 1985).

[20] Tsetlin, M.L., Automaton Theory and Modeling of Biological Systems (Academic Press, New York, 1973).

[21] Halpern, J.Y., and Fagin, R., A formal model of knowledge, action, and communication in distributed systems: preliminary report, in: Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing (Ontario, August 1985).

[22] Traub, J.F., Wasilkowski, G.W., and Wozniakowski, H., Information, Uncertainty, Complexity (Addison-Wesley, Reading, Massachusetts, 1983).

[23] Dechter, R. and Kleinrock, L., Broadcast Communications and Distributed Algorithms, Technical Report, Computer Science Dept., Univ. of California, Los Angeles, 1984. To appear in IEEE Trans. on Computers.

[24] Yemini, Y., A statistical mechanics of distributed resource sharing mechanisms, in: Proceedings of INFOCOM 83, (1983) 531-539.